VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Možnosti Unreal Enginu 4

Possibilities of Unreal Engine 4

2017 Bc. Jan Urubek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jan Urubek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Možnosti Unreal Enginu 4
Possibilities of Unreal Engine 4

Jazyk vypracování: čeština

Zásady pro vypracování:

Nová verze Unreal Enginu 4 sebou přináší pokročilé možnosti v oblasti vývoje her, vzdělávacích aplikací, vizualizací apod. To je velmi důležité zejména při simulací jevů a procesů, které nelze v reálnem světě jednoduše demonstrovat. Důvodem může být například to, že je to technicky nemožné nebo by přitom mohlo dojít k ohrožení zdraví skupiny nebo jedinců. Proto již dnes existují např. simulace operací v lékařství nebo simulace na úrovní molekulárního světa. Cílem této práce je zaměřit se na možnosti vizualizace v novém Unreal Engine 4. Práce se bude zabývat popisem a demonstrací vybraných části tohoto enginu a výsledky budou použity v aplikaci pro simulování reálného provozu virtuální jaderné elektrárny.

1. Nastudujte možnosti a techniky při návrhu a implementaci projektu pro realistickou vizualizaci v Unreal Engine 4.
2. V práci se zaměřte zejména na následující části:
a) Vizualizaci interiéru.
b) Modelování a vizualizaci terénu.
c) Animaci postav pomocí kostry (rigování).
3. Teoretické znalosti využijte k implementaci ukázkových příkladů pro zpracovávaná témata.
4. Příklady vhodně zakomponujte po domluvě s vedoucím do projektu pro vizualizaci jaderné elektrárny, popřípadě jiných vhodných projektů.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Němec, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. dubna 2017 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 1. dubna 2017 .....

Rád bych na tomto místě poděkoval Ing. Martinovi Němcovi, Ph.D. za užitečné rady ohledně tvorby práce, za jeho podporu a pozitivní přístup při konzultacích.

Abstrakt

Nová verze Unreal Enginu patří v současnosti mezi nejpoužívanější herní enginy a nabízí uživatelům pokročilé funkce jak v oblasti vykreslování 3D grafiky, tak i v dalších oblastech jako fyzikálních simulací, audiovizuální tvorby, programovaní herní logiky či vytváření složitých scenérií. Proto byly v této práci prozkoumány specifické možnosti tohoto enginu, konkrétně témata jako: skeletální animace postav, vizualizace terénu či vizualizace interiéru.

Pro lepší nastínění problematiky, se každá z kapitol skládá z popisu problému, definic potřebných pojmů a konečného popisu postupu práce při implementaci ukázkových aplikací. Spolu s projektem vizualizace jaderné elektrárny bylo vytvořeno celkově 7 demonstračních aplikací, včetně dvou hratelných VR her.

Klíčová slova: Unreal Engine, herní engine, 3D grafika, 3D animace, vizualizace terénu, vizualizace interiéru, materiály, inverzní kinematika, lightmap artefakty

Abstract

New version of Unreal Engine is currently one of the most used game engines in the world and offers advanced functionality such as 3D graphics rendering, physical simulations, audiovisual production, game logic programming or creation of complex scenes. Therefore this work focuses on examination of specific possibilities in this engine, such as: skeletal animations of characters, terrain visualization or interior visualization.

For better introduction to the described problem, each chapter contains its description, definition of required terms and final description of the sample applications implementation. Along with virtual power plant visualization, in total seven sample applications were developed, including two playable VR games.

Key Words: Unreal Engine, game engine, 3D graphics, 3D animations, terrain visualization, interior visualization, materials, inverse kinematics, lightmap artifacts

Obsah

Seznam použitých zkratek a symbolů

UE4	–	Unreal Engine 4
mesh	–	3D model
fps	–	z angl. First Person Shooter - střílecí hra z první osoby
SSAO,DFAO,Lightmass AO		techniky používané při aproximaci ambientní okluze 3D scény v reálném čase
LOD	–	z angl. Level of Detail - jedná se o úrovně složitosti daného objektu ve scéně. V praxi se tato technika používá při optimalizaci scény, kdy se postupně přepínají různě složité modely pro jeden konkrétní objekt v závislosti na jeho vzdálenosti od kamery.
CPU	–	z angl. Central processing unit - procesor (elektronická součást počítače)
GPU	–	z angl. Graphics processing unit - grafický procesor (elektronická součást počítače)
RAM	–	z angl. Random Access Memory - operační paměť počítače
FPS	–	z angl. Frames per Second - počet vykreslovaných snímků za sekundu
C++,JavaScript,Boo,C#		programovací/skriptovací jazyky
Blueprint	–	Systém pro vizuální skriptování logiky aplikace v Unreal Engine 4
shader	–	Počítačový program sloužící k řízení jednotlivých částí GPU
vertex	–	Bod v prostoru. Používá se v topologii 3D modelů.
quad	–	Stěna v 3D modelu obsahující 4 body (vertexy)
LBS	–	z angl. Linear Blend Skinning - algoritmus používaný pro výpočet změny pozice jednotlivých vertexů při použití skeletálních animací
HLSL	–	z angl. High-Level Shader Language - vyšší programovací jazyk určený pro psaní shaderů vytvořený společností Microsoft pro Direct3D API
VR	–	Virtuální Realita
trigger	–	Kolizní objekt sloužící ke spouštění událostí v aplikaci. (např. když se jej dotkne hráčova postava
CG	–	z angl. Central Graphics - vyšší programovací jazyk určený pro psaní shaderů vytvořený společností NVIDIA
UV unwrap	–	Proces rozbalení ploch 3D modelu do 2D textury.
AO	–	Okluze ambientního světla

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V poslední době se stávají herní enginy stále více populární, jak v komerční sféře u vývojářských firem, tak i u jednotlivců, jelikož je lze snadno získat. Jsou to komplexní nástroje, které nabízí rychlou implementaci herních mechanik, vykreslování složitých a rozsáhlých scén v reálném čase, podporu pro různé platformy (od mobilních operačních systémů po herní konzole) a mnoho dalších funkcionalit potřebných k vytvoření požadovaných aplikací. Jak jejich název implikuje, jsou nejvíce využívány v oblasti vývoje her, avšak mohou být použity i pro jiné účely, jako například pro vizualizaci vědeckých pokusů, testování nových prototypů ovládacího hardwaru či implementaci prezentačních aplikací.

Aktuálně existuje mnoho herních enginů, kde mezi nejpoužívanější patří: Unity3D, Unreal Engine, CryEngine, Source, GameMaker, atd. Alternativou může býti vlastně psaný engine, což sebou přináší mnoho výhod i nevýhod. Výhodou v tomto případě může být lepší optimalizace výsledné aplikace a naprostá kontrola jejího chování včetně možnosti implementace specifických vlastností enginu. Nevýhodou je však vysoká časová náročnost při implementaci.

Proto je vždy před začátkem vývoje vhodné zamyslet se zda výsledná aplikace nelze vytvořit pomocí předem vytvořených enginů, či zda je nutné začít psát vlastní engine.

2 Herní engine

Herní engine je software sloužící ke snadnému vývoji grafických aplikací, simulací a her. Jeho jádro umožňuje uživatelům využívat již naimplementovaných funkcionalit (jako vykreslování 3D a 2D grafiky, detekce kolizí, simulace fyziky, audio systém, atd.) a tudíž není nutné pro vývojáře tyto základní funkce implementovat znovu a znovu. [1]

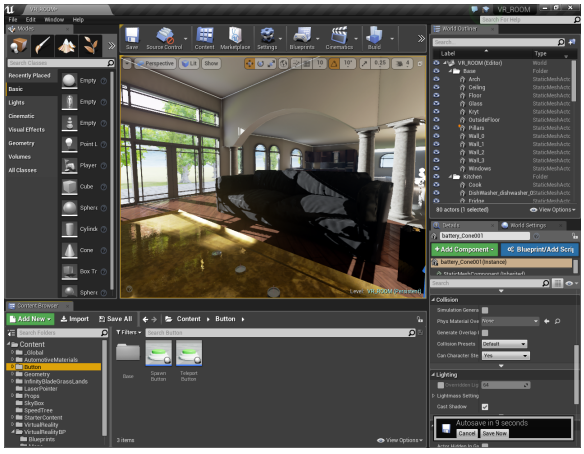
Úkolem uživatele tohoto enginu je implementace vlastních mechanik a naplnění projektu vlastním obsahem (3D modely, textury, zvuky, atd.).

2.1 Unreal Engine

Jedním z často používaných herních enginů je Unreal Engine vytvořený firmou Epic Games v roce 1998 pro 3D fps hru Unreal. Od té doby se engine neustále vylepšuje a využívá při vývoji 3D aplikací a her různých žánrů, ačkoliv byl původně tento engine směřován pouze k hrám typu fps. V současné době je k dispozici již čtvrtá verze, která je na velmi vysoké úrovni zejména díky neustálé snahy autorů o vylepšování a přidávání moderních funkcionalit. Jádro tohoto engine je napsáno v C++ a jeho zdrojové kódy jsou volně přístupné, což představuje výhodu oproti jiným enginům, jelikož se vývojář může podívat jak byla určitá funkce implementována.

Oproti svým předchozím verzím, Unreal Engine 4 obsahuje mnoho významných funkcí, jako například globální iluminace v reálném čase, fyzikálně založené materiály, vizuální programování herní logiky pomocí blueprintů, podpora DirectX11 a DirectX12, sofistikovaný systém vytváření a optimalizace při vykreslování rozsáhlých terénů s přidanou vegetací a post-process efekty jako: SSR (Screen Space Reflections), 3 druhy ambientní okluze (SSAO,DFAO,Lightmass AO) a mnoho dalších.[2]

Unreal Engine 4 je možno bezplatně stáhnout a využívat jej, avšak firma Epic Games si účtuje 5% z výdělku produktů, jejichž celkový výnos přesáhl 3000$.[2]



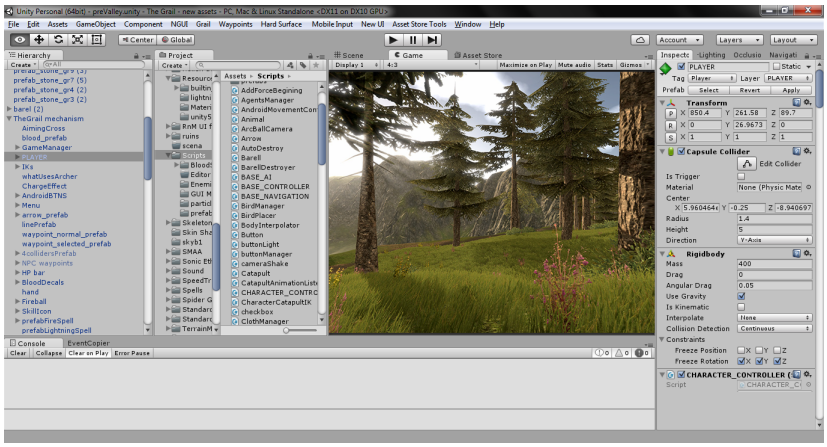
Obrázek 1: Rozhraní Unreal Engine 4

2.2 Unity3D

Unity3D je v současnosti nejpoužívanějším herním enginem. Zásluhou na tom má zejména jeho oblíbenost u vývojářů aplikací/her určených pro mobilní trh, jeho jednoduché a intuitivní rozhraní, rozsáhlá podpora cílových platforem (v současnosti podporuje 28 platforem - od PC po OS smart televizí), programování v jazyce C# či JavaSript, mnoho optimalizačních funkcí (Static batching, Dynamic batching, LODy, Occlusion Culling, atd.), fyzikálně založené stínování a globální iluminace, užitečný profiler monitorující zatížení jak CPU tak i GPU a RAM paměti, a mnoho dalších užitečných vlastností.

První verze Unity3D byla vydána v roce 2005 v jádře napsaném v jazyce C a C++. V současnosti je přístupná verze číslo 5, jež byla přelomová díky podpoře fyzikálně-založeného stínování a nového způsobu licencování uživatelů.

Tento engine je zcela bezplatný a používat jej může v jeho plné funkcionalitě kdokoliv. Výjimku tvoří ti uživatelé, jejichž profit z vydané hry vytvořené v Unity přesáhl 100000$. Tito uživatelé jsou povinni zakoupit tzv. Pro verzi v hodnotě 1500$.[3]



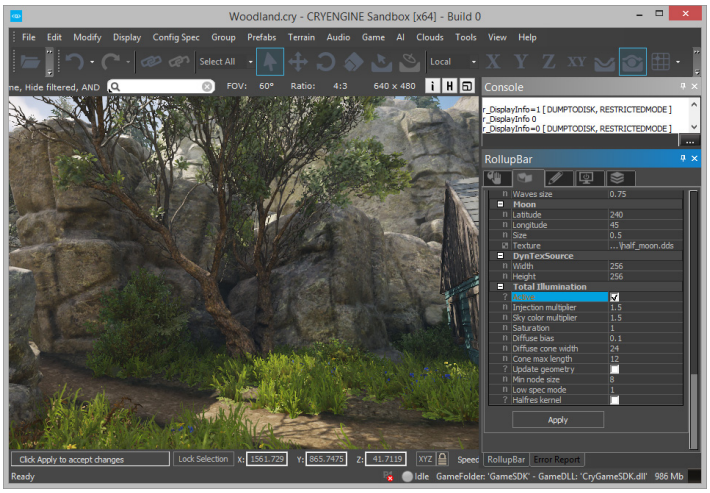
Obrázek 2: Rozhraní Unity 5

2.3 CryEngine

CryEngine je komplexní herní engine vytvořený německou firmou Crytek. Tato firma jej vyvinula původně jako technologické demo pro společnost NVIDIA, která v něm viděla velký potenciál a následně v něm byla vytvořena hra FarCry, jež ve své době představovala špičku herní grafiky. S následnými verzemi této hry vznikalo mnoho dalších graficky vyspělých her, zejména hry série Crysis, jež byla celosvětově známá a často využívána k testování výkonu grafických karet a měření FPS.

V současnosti je k dispozici již pátá verze CryEngine, která podporuje funkce jako: Voxel-Based Global Illumination (SVOGI), podpora DirectX 12, fyzikálně-založené stínování, Image based lightning, Volumetric Fog Shadows, a mnoho dalších špičkových efektů. Dále je zdrojový kód tohoto enginu volně dostupný pro členy CRYENGINE komunity, což stejně jako u Unreal enginu představuje výhodu pro vývojáře.

Dostupnost tohoto enginu je založena na "Pay what you want"modelu, tedy v překladu "Zaplaťte tolik, kolik chcete". Tudíž uživatel si jej může stáhnout i využívat bezplatně a poté se rozhodnout zda podpoří firmu Crytek zasláním finančního daru.[4]



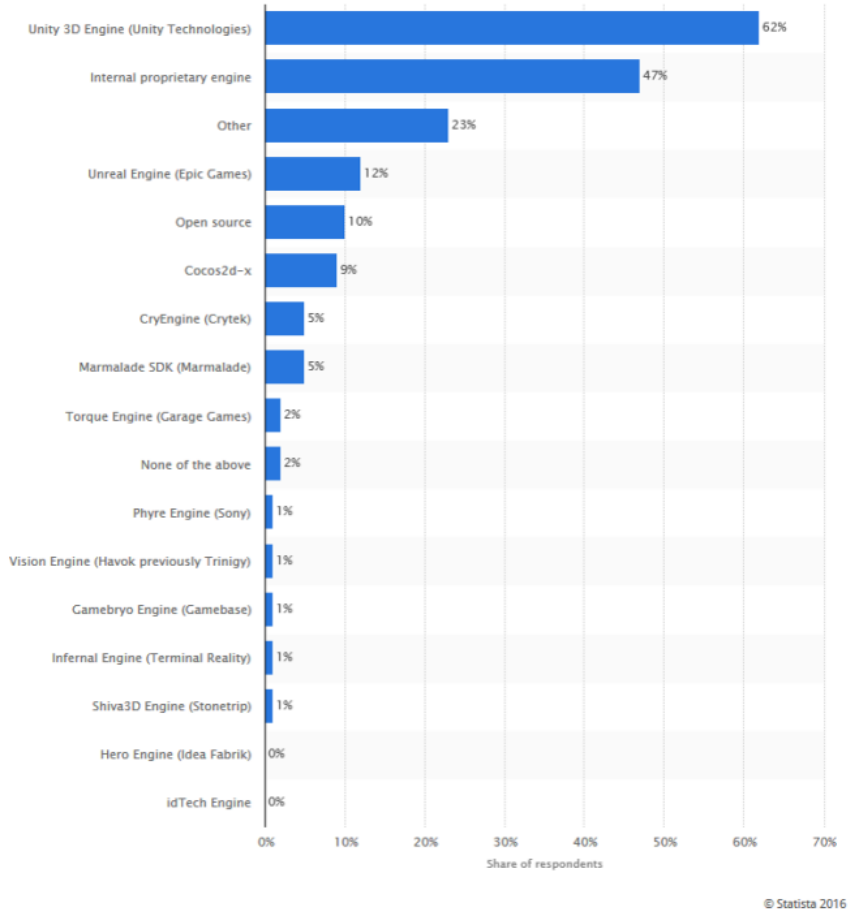
Obrázek 3: Rozhraní CryEngine 5 [4]

2.4 Srovnání Unreal Engine 4 a Unity3D

Jednou ze součástí této práce bylo stručné srovnání Unreal Engine 4 s dalším velice populárním enginem Unity3D. Toto srovnání se zaměřuje zejména na popularitu, postupy při implementaci logiky aplikace a tvorby materiálů/shaderů.

2.4.1 Popularita

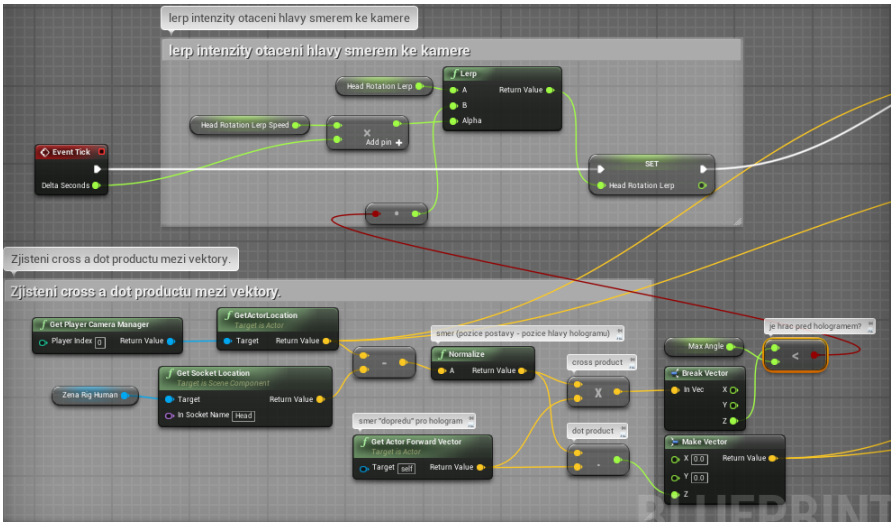
V oblasti popularity vede herní engine Unity3D. Jak lze vidět z ankety na obrázku 4, více než 60% z 10000 tázaných respondentů (vývojářů) používá právě Unity3D. Této oblibě napomáhá zejména vývoj aplikací pro mobilní trh, k němuž je Unity3D více uzpůsobeno, jak ve směru optimalizace, tak v rychlosti vývoje.



Obrázek 4: Výsledky ankety popularity herních enginů, zodpovězené více než 10000 vývojáři z Anglie[5]

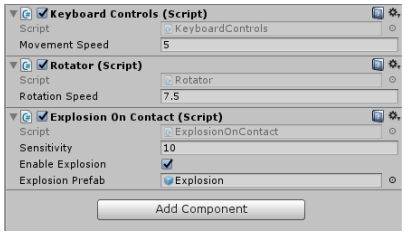
2.4.2 Implementace logiky

V předmětu implementace algoritmů herní logiky, je vizuální programování pomocí blueprintů v UE4 vhodné i pro uživatele bez hlubších programovacích znalostí. Pro pokročilejší vývojáře však existuje i možnost programovat logiku v jazyce C++. Základní třídou pro UE4 je třída Actor, jež představuje rodičovskou třídu pro každou entitu ve scéně.



Obrázek 5: Blueprint v UE4

Unity3D na rozdíl od UE4 využívá pouze programovacích jazyků jako C#, javascript či skriptovacího jazyku Boo (pouze ve starších verzích Unity). Tyto jazyky jsou použity při psaní tzv. skriptů, které definují chování určitého objektu ve scéně. Velikou výhodou oproti UE4 je možnost mít více různých skriptů (komponent) na jednom konkrétním objektu, což v UE4 není možné, jelikož o chování určitého objektu se starají pouze algoritmy daného Actora.

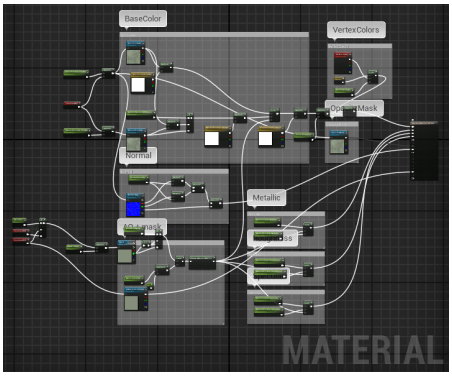


Obrázek 6: Přidávání skriptů k jednomu objektu v Unity

2.4.3 Materiály/Shadery

V oblasti vytváření materiálů, nabízí Unreal Engine tzv. Material Editor, což je systém velmi podobný systému blueprintů, tedy vizuální spojování funkčních uzlů. Práce v tomto editoru je velmi přehledná a uživatel dokáže pozorovat vlivy různých funkčních uzlů na vzhled materiálu v reálném čase, díky funkcím **Live Preview** a **Live Update**. Také je zde možnost napsání vlastníhch funkcí v jazyce HLSL a přidání do materiálu za pomocí uzlu **Custom** [18].

Pro každý vytvořený materiál lze vytvořit jeho instance, což je v podstatě jednoduchá forma jeho duplikování za pouhým účelem změny vstupních parametrů (textur, skalárních hodnot, vektorových hodnot, atd.). Lze tedy vytvořit jeden rodičovský materiál, ze kterého budou dědit veškeré instance použité ve scéně.



Obrázek 7: Editor materiálů v UE4

Unity (základní verze bez přidaných funkčních assetů) na druhou stranu používá pouze textovou formu psaní shaderů, pomocí různých maker a využití Cg jazyku. Takto napsané shadery se přiřadí daným materiálům a uživatel poté pouze mění jejich vstupní parametry. [17]

```
void surf (Input IN, inout SurfaceOutputStandard o) {
  fixed4 c = tex2D(_MainTex, IN.uv_MainTex) * _MainColor;
  fixed4 em = tex2D(_Emmisive, IN.uv_MainTex) * _EmmisiveIntensity;

  o.Albedo = c.rgb;
  o.Alpha = c.a;
  o.Metallic = _Metallic;
  o.Smoothness = _Glossiness;
  o.Normal = UnpackNormal(tex2D(_BumpMap, IN.uv_MainTex));

  half rim = 1 - saturate(dot(normalize(IN.viewDir), o.Normal));
  o.Emission = em.rgb + (_OutlineColor.rgb * pow(rim, _Outline));
}
```

Výpis 1: Ukázka výpočtu barvy v jazyce Cg

Při porovnání těchto dvou způsobů má Unreal Engine uživatelský příjemnější prostředí a přehlednější zobrazení postupu výpočtu výsledné barvy daných materiálů.

3 Skeletální animace postav

Animace postav bezpochyby patří mezi často využívané funkce herních enginů. Pro tento byly vytvořeny tzv. armatury pomocí kterých je možno deformovat určené části modelu postav, díky čemuž vznikají animace.

- **Skinning** - Pojmem skinning ve 3D grafice, je myšlena změna pozic bodů (vertexů) dané topologie 3D modelu, za pomoci tzv. kostí. Samotná deformace topologie je provedena za pomoci transformací (translation,rotation,scale) těchto kostí, což má za důsledek změnu pozic právě těch vertexů, jež jsou k dané kosti "přichyceny"za pomocí vah. Pojem váha kosti k danému vertexu je skalární parametr v rozmezí 0-1, kdy hodnota 1 reprezentuje maximální váhu a 0 minimální (žádnou) váhu. Jeden vertex však může být ovlivněn více kostmi naráz, kde každá kost může mít jinou váhu, a tudíž se výsledná pozice vypočítá s ohledem na všechny ovlivňující kosti. Jedním z nejpoužívanějších algoritmů výpočtu je algoritmus LBS (z anglického Linear Blend Skinning)[6].

  Rovnice pro výpočet nové pozice vertexu, podle algoritmu LBS :

$$\mathbf{v'_i} = \sum_{j=1}^{m} w_{i,j} T_j v_i \tag{1}$$

v'	-	transformovaná (výsledná) pozice vertexu ve world space
v	-	základní (výchozí) pozice vertexu v local space
w	-	váha j-té kosti pro i-tý vertex
T	-	transformační matice j-té kosti

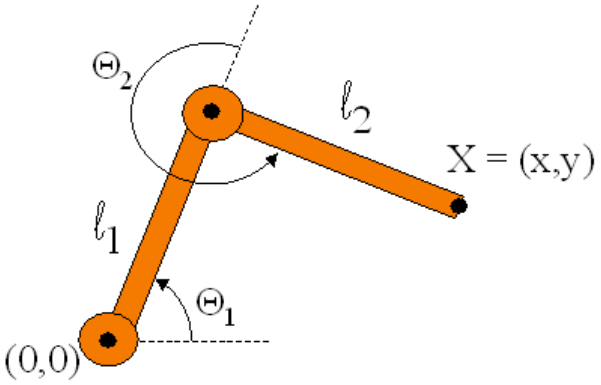
- **Armatura** je hierarchická stromová struktura kostí. Proces jejího vytváření a následného přiřazování vah jednotlivých kostí k jednotlivým vertexům se nazývá **rigování**.

- **Póza** je jeden záznam o transformacích (většinou všech) kostí armatury v jeden okamžik.

- **Animace** představuje proces zaznamenávání jednotlivých póz armatury (transformací kostí) do určitých časových úseků (keyframů). Časová prodleva mezi jednotlivými keyframy není uniformní a tudíž si animátor může nastavit požadovanou rychlost animace. Velkou výhodou během animování postav je interpolace mezi po sobě jdoucími pózami a možnost vybrat si určitý typ profilu této interpolace (lineární, kvadratická, Bézierova křivka, atd.). Díky tomu se animátorovi výrazně ušetří čas, jelikož nemusí zaznamenávat pózy pro všechny keyframy, ale jen pro ty, ve kterých nastane důležitá změna v póze animované postavy.

3.1 Inverzní kinematika a praktické použití při řešení problému

Inverzní kinematika (anglicky inverse kinematics) je často využívaná technika pro výpočet správných hodnot úhlů natočení jednotlivých spojujících kloubů v řetězci tak, aby co nejpřirozenějším způsobem kopírovaly způsob ohybu reálných končetin. Přičemž jsou známy pozice prvního a posledního kloubu v řetězci. Tato technika je často využívaná jak v robotice, při řešení problému pohybu robotického ramena, tak i v oblasti počítačové grafiky, kdy je potřeba animovat přirozené chování kloubů u postav. Dobrým příkladem může být správné našlapávání postav při vycházení schodů či různých nerovností terénu.



Obrázek 8: Výpočet úhlů kloubů pomocí inverzní kinematiky[7]

Z obrázku výše lze vidět, že při výpočtu úhlů Θ_1, Θ_2 je nutné znát délky jednotlivých kostí l_1, l_2 a pozice prvního a posledního kloubu v řetězci. Hledané úhly Θ_1, Θ_2 lze vypočítat pomocí rovnic 2 a 3 [7].

$$\Theta_2 = cos^{-1}\left(\frac{x^2 + x^2 - l_1^2 - l_2^2}{2l_1 l_2}\right) \tag{2}$$

$$\Theta_1 = \frac{-(l_2 sin(\Theta_2))x + (l_1 + l_2 cos(\Theta_2))y}{(l_2 sin(\Theta_2))y + (l_1 + l_2 cos(\Theta_2))x} \tag{3}$$

3.1.1 Praktické použití

Pro otestování inverzní kinematiky byl řešen problém animace postavy, reprezentující samotného hráče ve VR. Model postavy hráče byl tedy ovládán třemi kontrolními body : 2 ručními ovladači a VR brýlemi HTC Vive. Jelikož k decentní animaci je potřeba více kontrolních bodů, bylo nutno výslednou rotaci kostí aproximovat pomocí inverzní kinematiky, vektorových operací a mixování několika předpřipravených animací.

Základem bylo animování a vytvoření přechodů mezi čtyřmi typy animací : stání, klečení, úkroky do strany a úkroky dopředu. V tomhle ohledu bylo za potřebí zaznamenávat dva 3D

vektory reprezentující aktuální pozici a minulou pozici headsetu. Mezi těmito vektory byl vypočítán za pomocí odčítání pohybový vektor, jež byl často využíván v animačním blueprintu hráče.

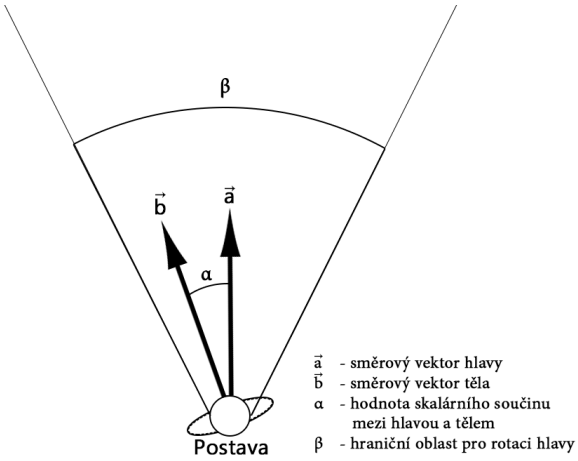
Obrázek 9: Ukázky aproximace hráčovy postavy ve virtuálním zrcadle

Výsledný animační blueprint byl vcelku komplexní a obsahoval mnoho nezbytných funkcí. Jednou z nich byla detekce pohybu v negativním směru, jež se vypočítal pomocí skalárního součinu mezi vektorem pohybu a forward vektorem hráčovi hlavy. Další funkce zajišťovaly jak detekci mezi chůzí a stáním (pomocí délky vektoru pohybu), tak detekci mezi klečením a stáním (pomocí pozice headsetu v Z-tové rovině).

Pro vytvoření aproximované rotace celého těla a hlavy hráče, bylo nutno navrhnout jednoduchý algoritmus, který se řídil těmito pravidly:

Natoč hlavu hráče stejným směrem jako rotace reálné kamery;
Zjisti směrové vektory $\vec{a}, \vec{b}$ // $\vec{a}$ - *"forward"vektor hlavy,* $\vec{b}$ - *"forward"vektor těla*;
Vypočítej skalární součin α mezi vektory $\vec{a}, \vec{b}$;
if $\alpha < 0.75$ **then**
| Zahaj interpolaci rotace těla ve směru rotace hlavy podle osy Z ("yaw")
else
| Jinak tělem nerotuj
end

Algoritmus 1: Algoritmus rotace těla



Obrázek 10: Vizualizace algoritmu 1

Rovněž bylo nutno neustále v blueprintu hráče aktualizovat proměnné reprezentující pozice a rotace všech tří kontrolujících zařízení. Tyto proměnné se poté synchronizovali v animačním blueprintemu, kde sloužili jako kontrolní body pro funkce inverzní kinematiky. Tyto funkce byly použity třikrát, tedy pro každé kontrolní zařízení. Dvě pro aproximaci pohybu rukou a jedna pro aproximaci sklánění hráče v zádech.

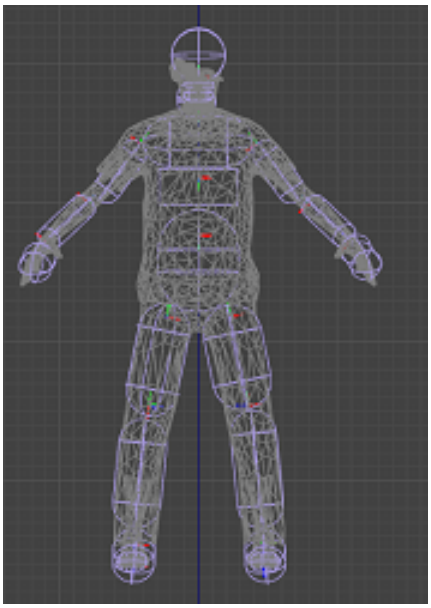
Ve výsledku byly pozice rukou i hlavy dobře zesynchronizovány s reálnými pozicemi testujícího hráče. Problém však představovalo natočení rukou (loktů) hráče, jelikož pro správnou funkčnost potřebuje inverzní kinematika ještě jeden kontrolní bod, reprezentující bod, ke kterému loket směřuje. Pro tyto kontrolní body neexistovali žádná reálná data a tudíž jim byla nastavena fixní pozice za zády hráče, což v některé situaci představovalo problém (například při rozpažení).

Obrázek 11: Ukázka nesprávné rotace kostí

Tento problém by mohl být vyřešen komplexním algoritmem, jež by modifikoval pozice těchto kontrolních bodů, v závislosti na lokální pozici a rotaci ručních ovladačů. Ovšem i v tom případě by se jednalo o aproximaci a ve výsledku by nebylo možné vytvořit zcela přirozené animace hráče, kvůli nedostatku kontrolních zařízení.

3.2 Fyzikální animace hráče

Běžné animování postav se skládá z ukládání vytvořených póz do časových úseků. Tento způsob je vhodný pro základní animace jako chůze, běhání, stání, útok, atd. Avšak pro situačně spjaté animace, jako je například smrt nepřítele a pád jeho těla určitým směrem s ohledem na kolizní objekty ve scéně, bylo nutné použít tzv. fyzikální animace. Základem těchto animací je vytvořit vybraným kostem jejich kolizní modely a nastavit jejich fyzikální vlastnosti. Poté je nutné vytvořit mezi těmito kolizními modely klouby (tzv. jointy). U těchto kloubů lze editovat mnoho proměnných, které ovlivní chování ohybu těchto kloubů. Například lze zafixovat rotace v požadovaných lokálních osách, či vytvoření hraničních hodnot pro tyto rotace (angular limits), což je vhodné pro simulování přirozeného chování například ohybu koleního kloubu. Samotná aktivace fyzikální animace v průběhu hry se provede pomocí uzlů **Set simulate physics** a **Wake all rigid bodies**.



Obrázek 12: Zobrazení kolizních modelů

Obrázek 13: Nasimulovaný pád

Pro práci s kolizními modely a editaci vlastností kloubů v UE4 existuje speciální integrovaný editor s názvem Physics Asset Tool (PhAT), v jehož prostředí si uživatel může buď ručně nebo automaticky vytvořit fyzický model (ragdoll) pro specifikovaný mesh postavy.

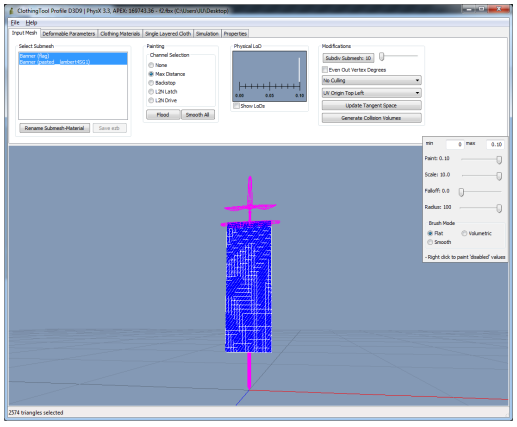
3.3 APEX Clothing - simulace látky

Realisticky vypadající animace látky vždy zvyšuje vizuální kvalitu výsledné aplikace. Z tohoto důvodu byla v této práci prozkoumána možnost využití technologie APEX Clothing od firmy NVIDIA [15], jež podporuje právě realistickou fyzikální simulaci látky v reálném čase.

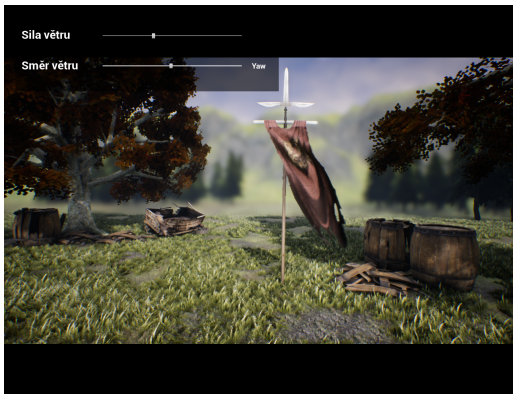
Pro vytvoření ukázkové aplikace této technologie, byl nejprve vymodelován jednoduchý model reprezentující prapor. Tento mesh se skládal z dvou sub-meshů: držák a samotná vlajka. Důležitou poznámkou je, že pro správné fungování simulace látky, musela být pro tento model vytvořena jednoduchá armatura obsahující jedinou kost.

Dalším nutným krokem bylo získání APEX SDK, jehož součástí byl program **ClothingTool Profile** ve kterém se nastavilo chování látky daného modelu. Přesněji určilo se které vertexy budou fixní a které dynamické, jakou maximální vzdálenost od jejich počáteční pozice mohou dosáhnout, jaké jednoduché kolizní tvary ovlivňují simulaci, ztuhlost látky, atd. Konečným výstupem tohoto programu byl soubor s příponou **.apx**, jež obsahoval výše uvedené parametry dané látky.

Posledním krokem bylo naimportování modelu vlajky do UE4. Typ importu modelu byl nastaven jako skeletal mesh, což je pro funkci simulace látky nutné. Následně v záložce Clothing, jež se nachází v nastavení modelu, bylo nutné nahrát vytvořený .apx soubor a přiřadit jej pro sub-mesh obsahující materiál látky. Pro simulaci větru byl do scény přidán **Wind Directional Source**, jež automaticky ovlivňuje chování všech látek ve scéně.



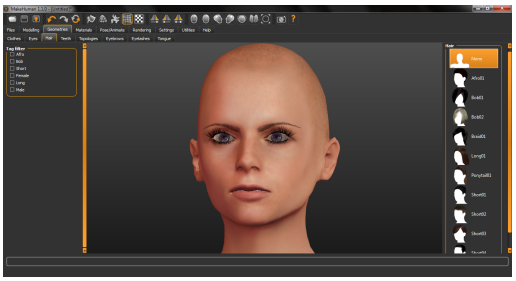
Obrázek 14: ClothingTool Profile



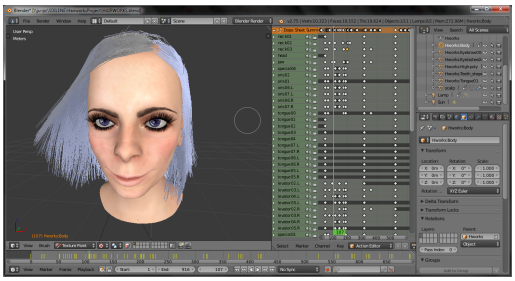
Obrázek 15: Výsledná látka v UE4

3.4 Animace obličeje a vlasů (NVIDIA HairWorks)

Další částí této práce bylo vytvoření animovaného modelu ženské tváře. Pro tento účel byl využit bezplatný nástroj MakeHuman[13], jež poskytuje jednoduché vygenerování modelu člověka pomocí grafického rozhraní. Součástí je i předdefinovaná armatura včetně základních kostí reprezentujících základní mimické svaly za pomoci kterých byly vytvořeny animace emocí.



Obrázek 16: Model v MakeHuman



Obrázek 17: Animování v Blenderu

Dalším nutným krokem bylo integrování technologie NVIDIA GameWorks do Unreal Enginu, jejíž součástí je právě i technologie HairWorks, která poskytuje kvalitní simulaci vlasů v reálném čase. Bylo tedy nutno obdržet zdrojové kódy UE4 s již integrovanými funkcemi GameWorks a zkompilovat je [14].

Následně byl v nové zkompilované verzi Unreal Engine vytvořen ukázkový projekt do kterého byl naimportován předešle vytvořený model ženské tváře i s animacemi. Pro přichycení vlasů, bylo nutno vytvořit blueprint jež obsahoval skeletalní mesh ženské tváře a **HairWorks scene component** jež reprezentoval samotné vlasy. Nakonec bylo vytvořeno i jednoduché rozhraní ovládající směr a sílu větru, či úhel pohledu hlavní kamery.

Obrázek 18: Ukázka HairWorks

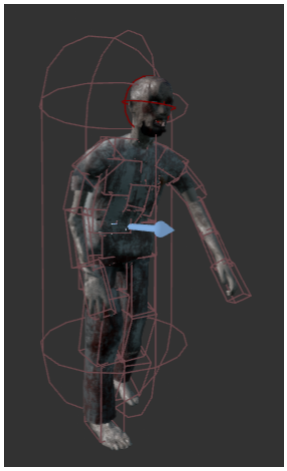
3.5 Dynamické odpojení částí nepřátelských těl

Tato podkapitola se zaměřuje praktickou implementací dynamického odpojení nepřátelských těl. Rovněž stručně popisuje následnou optimalizaci, při vykreslování velkého počtu těchto nepřátel.

3.5.1 Implementace

Prvním krokem bylo rozdělení meshe postavy na 30 částí pomocí modelovacího nástroje Blender a následně vytvoření třech úrovní detailů (LODů) pro každou část. Všechny tyto části byly poté naimportovány do UE4 jako skeletal meshe a vloženy do nepřítelova blueprintu. Dalším krokem bylo zesychronizování animací všech těchto částí pomocí uzlu **Set Master Component** a následně vytvoření neviditelných kolizních kvádrů typu trigger pro jednotlivé části. Tyto triggery následně sloužili jako indikátory zásahu pro fyzikální engine.

Samotná střelba hráče byla následně provedena za pomocí uzlu **Multi sphere trace for objects**, který slouží jako pomyslný paprsek s nastavitelným poloměrem, jež se vyšle z hráčovi zbraně a zasáhne fyzikální objekty ve scéně, jako právě zmíněné kolizní triggery na nepřítelově těle.



Obrázek 19: Triggery připevněné na jednotlivých částech nepřítele

Pří implementaci této aplikace nastali hned dva problémy:

1) Během zásahu určité části nepřítelova těla bylo nutné zasaženou část odpojit. To ovšem mohlo představovat problém v situaci, kdy je nutné odpojit i další části, které jsou k odstřelené části přichyceny a rovněž by přirozeně přichyceny zůstaly, i po oddělení původní části (např. je zasažen loket, tak zbytek ruky zůstane k němu přichycen). Byla tedy vytvořena hierarchie všech částí, která tento problém vyřešila.

2) Dalším problémem bylo oddělení podpůrných částí, tedy nohou. Nepřátelský zombie by logicky nemohl nadále chodit, tudíž byla při zásahu této části aktivována fyzika (ragdoll) a nepří-

tel přirozeně spadl na zem. Po menší časové prodlevě byly přepnuty přehrávané animace z chůze na plazení a problém byl vyřešen.

3.5.2 Optimalizace

Dalším problémem jež byl objeven během testování byla časová náročnost této funkce, jelikož na jednoho nepřítele bylo nutno vykreslovat třicet individuálních skeletal meshů. Jednoduchým a efektivním řešením bylo přepínání mezi vykreslováním původního celkového meshe a vykreslováním všech částí. Jelikož v situaci, kdy nebyl nepřítel vůbec zasažen, nebylo vůbec nutné vykreslovat všechny tyto části, ale jen jeden celkový mesh. Zviditelnění těchto částí by nastala až v okamžiku, kdy by byl nepřítel poprvé zasažen.

Obrázek 20: Původní mesh

Obrázek 21: Části

Dalšími kroky v ohledu optimalizace bylo použití tří úrovní detailů (LODů) všech meshů a využití postupné dealokace mrtvých nepřátel, což mělo za důsledek snížení počtu animovaných vertexů pro postavy nacházející se ve velké vzdálenosti od kamery.

Následně byla nutná redukce počtu kostí armatury, jelikož velký počet kostí při velkém počtu postav má neblahý vliv na výkon. Odstraněné kosti byly zejména v oblasti prstů a obličeje nepřátelských postav, jelikož tyto části moc neovlivnily potřebný vzhled animací.

4 Vizualizace terénu

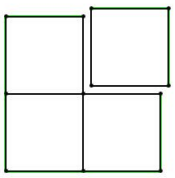
Pro vytvoření rozlehlých venkovních scenérií nabízí Unreal Engine komplexní nástroj pro vytváření a editaci terénu. V této kapitole je popsáno, jak je kalkulován celkový počet vertexů, postup generování nového terénu, aplikování materiálu a následně základní optimalizační funkce.

4.1 Rozlišení

Rozsáhlé mapy vyžadují i rozsáhlý terén, což znamená velký počet vertexů v topologii a problémy s tím spjaté z hlediska optimalizace. Kvůli tomu je terén rozdělen do n*m **komponent**, kde každá komponenta obsahuje buď 2*2 nebo 1*1 sekcí. Následně je každá tato sekce rozdělena na stejný počet quadů.

4.1.1 Komponenty

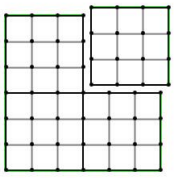
Každá z komponent je vždy čtvercovitého tvaru a má stejné rozměry jako ostatní komponenty. Tyto rozměry se vypočítají z celkové velikosti terénu a počtu komponent (jednoduše řečeno, komponenty rozdělí terén do rovnoměrné mřížky).



Obrázek 22: Jednoduchý terén rozdělen do čtyř komponent [8]

4.1.2 Sekce

Komponenty mohou být rozděleny do 1 nebo 4 (2x2) sekcí. Kde každá tato sekce obsahuje n^2 quadů (maximálně 256x256). Díky tomu je celkový počet quadů v celé komponentě buď $n^2 - 1$ (pokud je komponenta rozdělena na 1 sekci) nebo $n^2 - 2$ (pokud je komponenta rozdělena na 4 sekce).



Obrázek 23: Komponenta rozdělená do 4 sekcí, kde každá sekce je tvořena 9 (3x3) quady [8]

UE4 využívá tyto sekce jako základní jednotku terénu pro vypočítání a přepínání jednotlivých LODů.

4.1.3 Celkový počet vertexů

Před tvorbou nového terénu je vhodné se nejprve zamyslet, jaké by měl mít jeho celkové rozměry v porovnání s rozměry v reálném světě. Jelikož vzdálenost mezi dvěma vertexy tvořící hranu jednoho quadu je 1m, bude tedy celkový počet vertexů reprezentovat reálnou plochu terénu v m^2 (pokud je scale nastaven na *(1,1,1)*).

Tento celkový počet vertexů se dá snadno spočítat podle následujícího vzorce (2):

$$\mathbf{sum\_vert} = ((\mathbf{m_x} * (\mathbf{n} * \mathbf{s}) + \mathbf{1}) * ((\mathbf{m_y} * (\mathbf{n} * \mathbf{s}) + \mathbf{1}) \tag{4}$$

m_x - počet komponent (řádky).

m_y - počet komponent (sloupce).

s - rozlišení sekcí v komponentě (celkově je $s * s$ sekcí v komponentě)

n - rozlišení (počet quadů) v jedné sekci (celkově je $n * n$ quadů v sekci)

Při optimalizaci je nejdůležitější pokrýt co největší plochu a přitom použít co nejméně komponent. Pro jednodušší práci s nastavováním velikostí terénů, je zde vypsána tabulka doporučených rozlišení [8]:

Celkem vertexů	quady / sekce	sekce / komponenta	komponenty
8129x8129	127	4 (2x2)	32x32
4033x4033	63	4 (2x2)	32x32
2017x2017	63	4 (2x2)	16x16
1009x1009	63	4 (2x2)	8x8
1009x1009	63	1	16x16
505x505	63	4 (2x2)	4x4
505x505	63	1	8x8
253x253	63	4 (2x2)	2x2
253x253	63	1	4x4
127x127	63	4 (2x2)	1
127x127	63	1	2x2

Maximálním možným rozlišením terénu je 8129x8129, což představuje už velmi rozsáhlý terén. Avšak mohou nastat situace, kdy je nutné vytvořit mnohonásobně větší scenérie. Proto se nabízejí následující možností: **nastavení vyšších hodnot parametru scale** nebo **využití více terénů a level streamingu**.

První možnost je dvojsečná, jelikož při nastavení většího scale se terén skutečně zvětší, avšak ztratí požadované detaily, jelikož se zvětší vzdálenosti mezi jednotlivými vertexy. Což může způsobovat viditelně ostré hrany modelu terénu a nepřesnou kresbu jednotlivých vrstev textur, jelikož UE4 ukládá informace o vrstvě pro každý vertex. Proto může například nastat situace, kde se přechod mezi vrstvou trávy a hlíny nachází na nepožadovaném místě, jelikož chybí dostatečný počet vertexů pro určení přesnější polohy tohoto přechodu.

Mnohem lepším řešením je využití druhé možnosti, tedy level streamingu, při kterém se vytvoří větší počet terénů, kde je každý z těchto terénu obsažený ve vlastním levelu. Toto řešení má své výhody i nevýhody. Výhodou je zachování detailního modelu, jelikož mezery mezi jednotlivými vertexy nejsou nijak zvětšeny oproti předešle popsané technice. Rovněž je možné jednotlivé levely obsahující tyto terény postupně alokovat a dealokovat během hry, což sníží paměťovou náročnost výsledné aplikace. Nevýhodou na druhou stranu mohou být viditelně nenavazující okraje dvou terénů. Proto je vhodné využít profesionální programy pro tvorbu terénů, ve kterých lze vyexportovat dobře navazující výškové mapy pro každý z terénů. Nebo pokud to situace dovolí, je možno na viditelných místech zamaskovat tyto přechody různými objekty či vegetací.

Level streaming je tedy vhodným řešením při vizualizaci rozsáhlých map a pro jeho fungování UE4 využívá tzv. **Level Streaming Volumes** nebo manuálně vytvořených blueprintů obsahujících uzly **Load Stream Level** a **Unload Stream Level**, popřípadě funkce **UGameplayStatics::LoadStreamLevel** v jazyce C++.

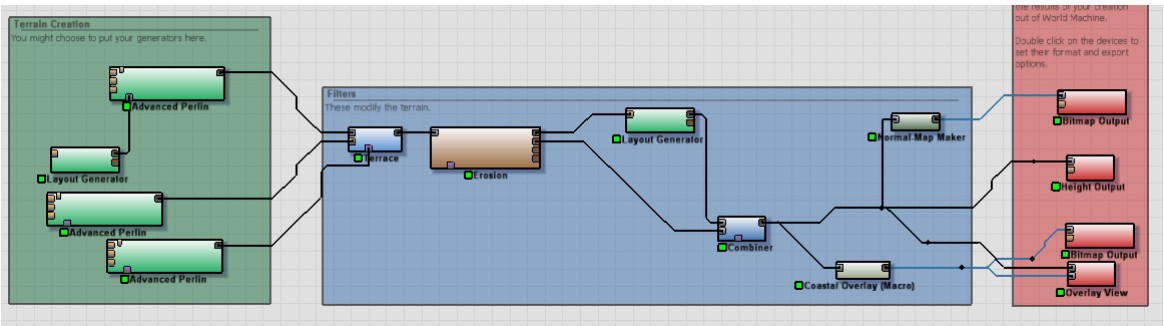
4.2 Materiály terénu a praktické použití

Součástí této práce byla praktická vizualizace terénu a vytvoření materiálů jež by napodobovaly reálné povrchy.

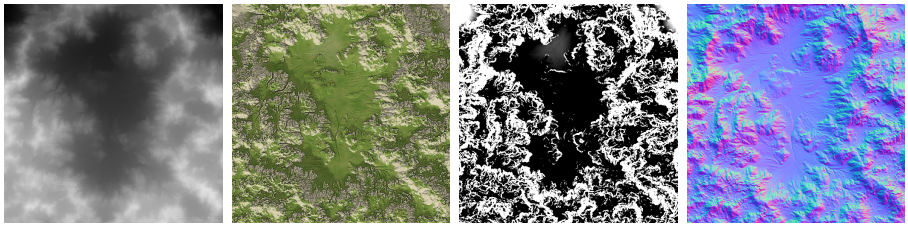
4.2.1 Vygenerování dat 3D terénu

Ačkoliv UE 4 disponuje nástroji pro editaci terénu, jedná se pouze o základní nástroje pomocí kterých lze velmi těžko napodobit nerovnosti reálných terénů. Což při praktickém použití znamenalo vcelku špatně vypadající a nereálné nerovnosti terénu. Z tohoto důvodu byl využit bezplatný a profesionální program World Machine[9], ve kterém byly vygenerovány specifikované výškové mapy a dalších data, za pomocí kombinace matematických funkcí a uživatelsky specifikovaným rozvržením terénu. Samotný postup tohoto generování je založen na vytvoření a propojení grafu (viz. obrázek 4.2.1), jehož uzly se dají rozdělit do skupin:

- **Vstupní uzly** - uzly které generují pixelová data, jsou to zejména matematické funkce (perlin noise, voronoi, gradient, atd.) nebo uživatelem vytvořená výšková mapa.
- **Filtry** - uzly které následně modifikují vstupní data (rozmazání, negativ, atd.)
- **Kombinující uzly** - uzly které zkombinují několik vstupních dat buď selektivně či pomocí matematické funkce (sčítání, odčítání, atd.)
- **Pokročilé funkce** - uzly obsahující pokročilejší funkce jako například: vytvoření normálové mapy, zapečení světelné mapy, aplikování eroze či dalších naturálních deformací terénu vlivem počasí, atd.
- **Uživatelské makra** - uživatelem vytvořené podgrafy funkcí, vnořené do jednoho uzlu
- **Výstupní uzly** - uzly které slouží k finálnímu vyexportování dat, které jsou typu: grayscale obrázek, RGB obrázek či 3D model. Rozlišení obrazových dat musí odpovídat počtu vertexů vytvářeného terénu (1 pixel = 1 vertex). Vhodné je použít jedno z rozlišení popsaných v tabulce 1 (sloupec "Celkem vertexů").



Obrázek 24: Vytvořený graf



Obrázek 25: Vyexportovaná data. Zleva : Výšková mapa, makro textura, maska kamení (splat mapa), makro normálová mapa

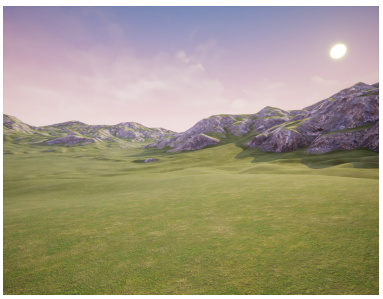
4.2.2 Stupně opakování textur

Za účelem reálnější vizualizace a redukce opakujících se vzorů textur na povrchu terénu, byl v praktické ukázce vytvořen materiál se třemi stupni opakování textur (tilingu). První stupeň reprezentoval povrch nacházející se blízko kamery a tudíž byl parametr **Mapping Scale** uzlu **LandscapeLayerCoords** nastaven na vysokou hodnotu, pro dosažení přirozeně vyšší hustoty texelů. Pro další stupně byl parametr **Mapping Scale** nastaven na nižší hodnoty, což postupně snižovalo hustotu texelů vzdálených ploch, díky čemu vynikly zejména textury skal.

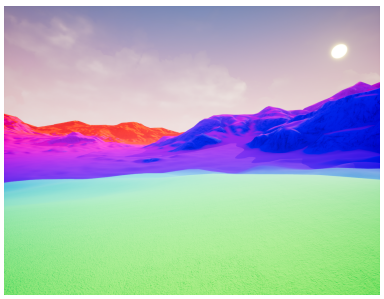
Dále byla pro všechny stupně využita textura **Perlinova šumu**, pro přidání větší variace odstínů. A také byla využita **makro textura** terénu, která se přimýchala k výsledné barvě pomocí lineární interpolace, za účelem realističtějšího vzhledu z ptačí perspektivy.

Samotný přechod mezi jednotlivými stupni byly vypočítán pomocí interpolace na základě vzdálenosti mezi kamerou a právě vykreslovaným fragmentem.

Obrázek 26: Bez stupňů



Obrázek 27: Se stupni



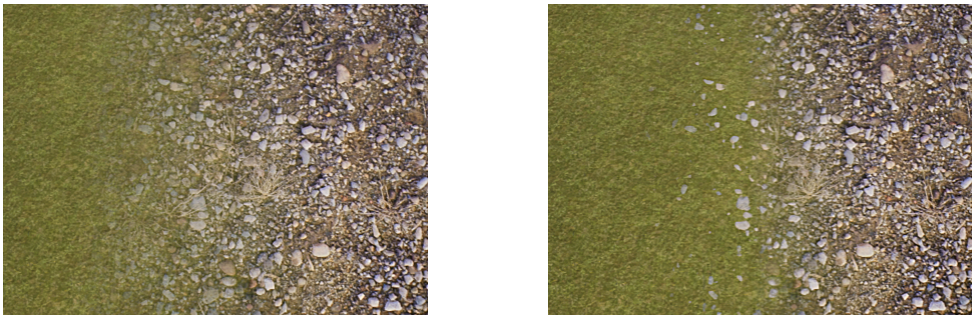
Obrázek 28: Přechody

4.2.3 Blending

Pro rozlišení určitých typů povrchu (tráva, bláto, atd.) je nutné kombinovat různé typy textur a dalších parametrů pomocí tzv. blendingu. V UE4 existují tři typy blendingu vrstev terénu a to:

- **LB Weight Blend** - Tento mód nerozlišuje pořadí jednotlivých vrstev, je tedy možno kreslit vrstvy nezávisle na ostatních.
- **LB Alpha Blend** - Tento mód rozlišuje pořadí jednotlivých vrstev, je vhodné jej použít když uživatel chce jasně určit která vrstva bude překrývat jinou vrstvu (například vrstva kamení vždy překryje trávu).
- **LB Height Blend** - Tento mód je stejný jako **LB Weight Blend** s tím rozdílem, že pro určitou vrstvu příjmá na vstupu další černobílou texturu určující maskování přechodu.

Obecně je vhodné použít pro první vrstvu **Alpha Blend** a pro ostatní **Height Blend** nebo **Weight Blend**. Jelikož může nastat vznik černých artefaktů v situaci kdy žádná vrstva nemá **Alpha Blend**, právě místech přechodu vrstev. **Alpha Blend** tedy zajistí, že první vrstva je vždy pod všemi ostatními vrstvami.

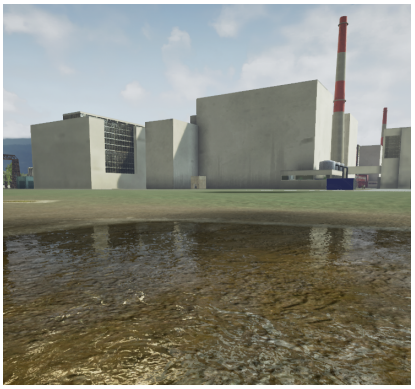


Obrázek 29: Přechody mezi vrstvami. (vlevo - Weight Blend, vpravo - Height Blend)

4.2.4 Vrstvy

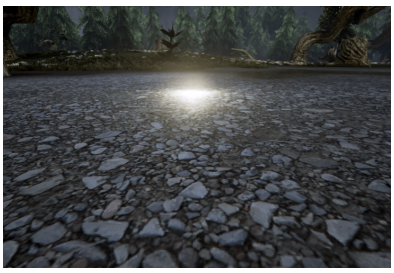
V praktické implementaci venkovní scény jaderné elektrárny bylo zapotřebí vytvořit povrchovou vrstvu kaluže/bláta, což představovalo trochu složitější úkol, jelikož má jiné vlastnosti z hlediska fyzikálně založeného stínování, oproti jiným vrstvám jako jsou tráva či hlína. Tato vrstva byla vytvořena kombinací dvou normálových map. První mapa je pouze čistá modrá barva, která reprezentuje rovnou hladinu kaluže vody. Druhá mapa je již samotná normálová mapa bláta. Tyto dvě mapy bylo nutno vhodně kombinovat za účelem vytvoření částečných nerovností bláta, které vystupují z vodní hladiny (z kaluže). Zároveň bylo nutno měnit hodnoty parametrů **metallic** a **roughness** pro každou z těchto map. Tudíž místa, ve kterých je voda, mají nízkou hodnotu

roughness a vysokou hodnotu metallic. Díky tomu vznikne částečná reflexe okolí.

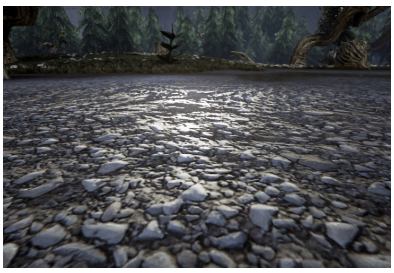


Obrázek 30: Materiál bláta (kaluže)

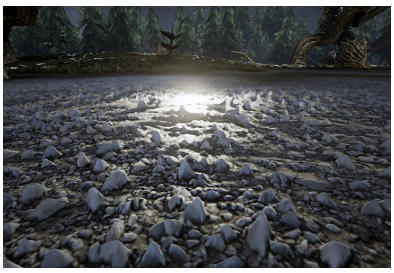
Další součástí bylo prozkoumání několika typů stínování a prakticky je aplikovat na vrstvu reprezentující povrch štěrku, jelikož obsahuje mnoho nerovností a správné stínování je nutností k lepšímu vzhledu terénu. V tomto ohledu bylo zaručeně nejvhodnější využít **Parallax Occlusion Mapping**[11], jelikož tato technika poskytuje věrohodnotnou iluzi nerovností štěrku.



Obrázek 31: Base



Obrázek 32: Normal



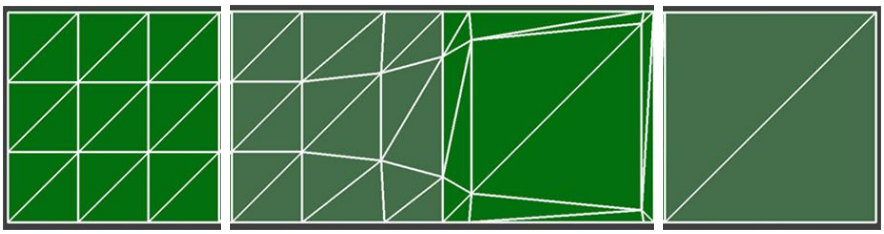
Obrázek 33: Parallax occlusion

4.3 Optimalizace

V ohledu náročnosti na prostředky se jedná o velmi optimalizovaný nástroj, a proto je vhodnější jej využít pro terén, nežli použít předem modelovaný statický mesh. Důvody jsou následující:

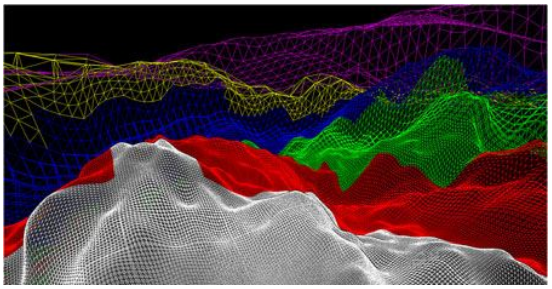
- **Nižší nároky na paměť** - terén využívá pouze 4 bajty na jeden vertex terénu, kdežto statický mesh používá 24-28 bajtů (12 bajtů pro pozici, 8 bajtů pro vektory tangentů a 16-bitové či 32-bitové UV koordinaty)
- **Geo-MipMap LODy** - přechod mezi jednotlivými úrovněmi detailů (LODy) je řešen pomocí standartních mipmap. Každá mipmapa reprezentuje jednu úroveň detailů, což ve výsledku dovoluje vysoký počet těchto úrovní a rovněž plynulý přechod, jelikož každá

dvojice mipmap může být vzorkována a následné odchylky v pozicích vertexů interpolovány ve vertex shaderu.

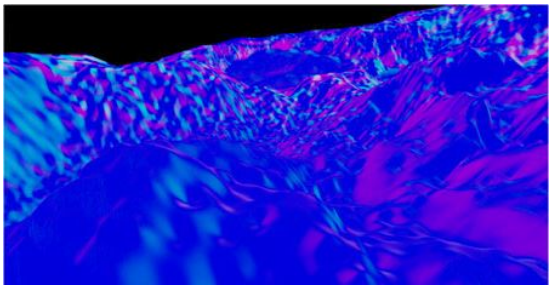


Obrázek 34: Přechod mezi LODy[10]

- **Data streaming** - jelikož veškerá data jsou uchovávána jako textury, je možno za běhu hry dealokovat z paměti ty textury daných LODů, jež nejsou zrovna vykreslovány a v situaci kdy je potřeba vykreslit, načtou se z disku do paměti.
- **Kvalitní osvětlení bez ovlivnění LODy** - jelikož veškeré nerovnosti povrchu lze zjistit z uložených dat (výškové mapy), jsou proto k dispozici data normálových vektorů ve vysoké kvalitě, jež není nijak ovlivněna nízkým počtem vertexů u vzdálených vrstvách LODů. V kombinaci s normálovými mapami povrchu terénu, vzniká velmi kvalitní osvětlení pro všechny úrovně LODů.



Obrázek 35: LODy terénu[10]



Obrázek 36: Kvalitní normály[10]

- **PhysX kolize** - terén v UE4 využívá speciální PhysX objekt založený na výškové mapě. V praxi to znamená menší nároky na výkon CPU, jelikož kolize s tímto objektem jsou počítány optimalizovanějším algoritmem, než kolize s fyzikálním objektem vytvořeným ze statického meshe. Rovněž je možnost pro tento speciální fyzikální objekt snížit rozlišení výškové mapy, což znamená další zvýšení výkonu s nadále přijatelnými kolizemi.

5 Vizualizace interieru

Poslední dobou je Unreal Engine 4 často využíván i v komerční sféře pro realistickou vizualizaci interiérů a různé architektonické projekty, bylo tedy jako další součást této práce prozkoumaní možností vizualizace interiérů, konkrétně dvou scén: podlaží blokové dozorny jaderné elektrárny a obývací místnosti propojené s kuchyní.

V této kapitole je popsán teoretický popis osvětlení, postup praktické realizace a řešení problémů.

5.1 Problémy

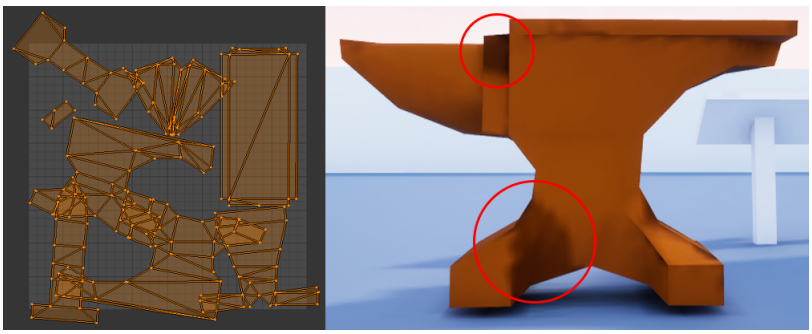
Součástí této práce bylo vytvoření aplikace určené pro demonstraci základních problémů vznikajících po zapečení lightmap. Tyto chyby byly znázorněny jak na modelech reprezentujících stěny interiéru, tak na modelech reprezentujících jednoduché objekty jako krychle či kovadlina. Technické příčiny a řešení těchto problémů budou rozebrány v této kapitole.

5.1.1 Překrývající se UV plochy

Jedním z poměrně častých problémů při zapékání lightmap jsou překrývající se UV plochy. Tento problém nastává pří nesprávném rozbalení (unwrappingu) ploch daného 3D modelu (např. zdi), když se dvě rozdílné plochy překrývají a při následném počítání osvětlení se přepíšou data již vypočítaného osvětlení pro stěnu A daty stěny B, a tudíž je osvětlení na stěně A nesprávné.

Během realizace tento problém několikrát nastal a jeho řešení bylo vždy poměrně jednoduché a přímočaré. Pro lightmap kanál (UV kanál č. 1) bylo nutno v modelovacím nástroji rozbalit (unwrap) všechny plochy modelu tak, aby se žádné stěny nepřekrývaly a aby jejich plocha nepřesahovala rozmezí 0-1.

Unreal Engine také nabízí automatické vygenerování UV map určených k lightmappingu, avšak tato metoda není vždy precizní a mnohdy při jejím použití nastává problém právě s překrývajícími se plochami. Proto je vždy lepší 3D modely rozbalit ručně.

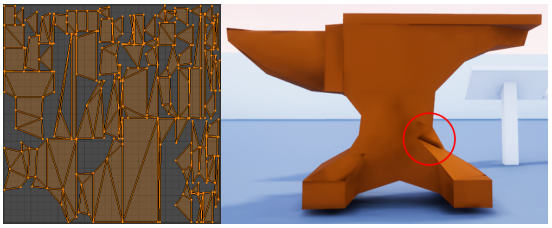


Obrázek 37: Překrývající se UV plochy

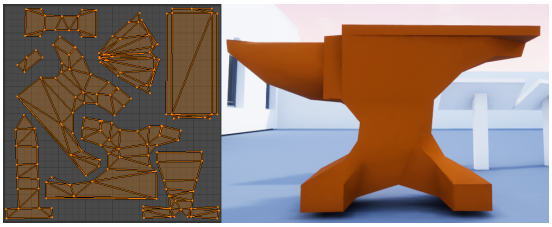
5.1.2 Bleeding

Dalším z častých problému je tzv. lightmap bleeding. Jedná se o problém, který rovněž nastává při nesprávném rozložení UV ploch modelu. Přesněji řečeno v okamžiku, když dvě rozdílné plochy nemají dostatečně velkou mezeru mezi sebou, což má za následek "prosvítání"zapečeného světla stěny B na stěnu A, čímž vznikají viditelné artefakty v oblasti hran.

Řešení tohoto problému je trochu složitější než u překrývajících se UV, jelikož je nutné při rozbalování ploch modelu dodržovat zásady jako je snaha o vytvoření co nejmenšího počtu rozbalených a spojených UV ploch, dodržování dostatečných mezer mezi plochy a zaplnění co největšího místa v textuře . Proto je pro daný model nutné v modelovacím nástroji vytvořit řezy (seams) takovým způsobem, aby při následném rozbalení vzniklo co nejmenší počet ploch, které mají konzistentní hustotou texelů nebo hustotu zvýšenou pro celkově viditelnější plochy a naopak sníženou pro ty plochy, které nejsou ve výsledku tolik důležité.



Obrázek 38: Špatné UV a bleeding

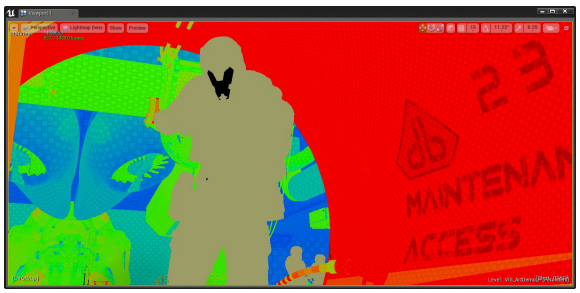


Obrázek 39: Správné UV bez bleedingu

5.1.3 Rozlišení lightmap

Rozlišení lightmap hraje velkou roli při kvalitě zapečeného osvětlení, zejména při zapékání statických stínů. Nastavení jeho velikosti se odvíjí od velikosti plochy modelu a uživatelem požadované kvality (např. pro modely reprezentující pozadí scény je vhodné použít nižší rozlišení, jelikož se k nim uživatel nedostane blízko).

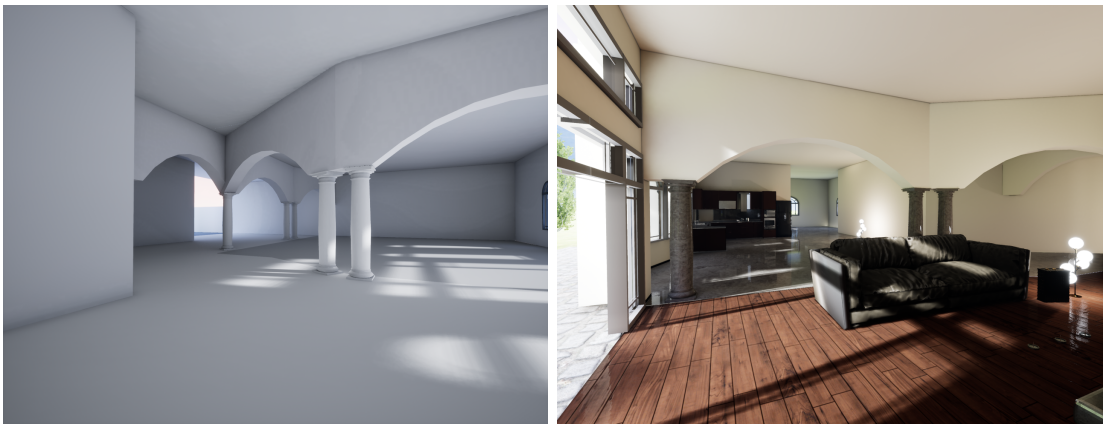
Pro nastavení správné velikosti rozlišení, poskytuje Unreal Engine mód zobrazení pojmenovaný **Lightmap density**, pomoci kterého uživatel snadno určí, které modely potřebují toto rozlišení zvýšit a které naopak snížit.



Obrázek 40: Mód zobrazeni Lightmap Density.
(modrá - nízká hustota texelů, zelená - ideální hustota textelů, červená - vysoká hustota texelu)[2]

5.2 Venkovní zdroje světla - lightmass portaly

Základem realistické vizualizace interiéru je správné nastavení osvětlení. Proto byl pro jednu z ukázkových aplikaci zvolen venkovní zdroj světla vytvořený za pomoci **Directional Light** a **Sky Light**, jež vstupovaly do scény pomocí oken a dveří. Po zapečení světel pro tenhle typ scény však nastal problém s grafickými artefakty na výsledných texturách lightmap. Řešení spočívalo v použití tzv. **Lightmass portálů**, což jsou v podstatě objekty určující místa, kterými vstupuje světlo do scény. Díky tomu může lightmapovací algoritmus určovat směr paprsků vysílaných z počítaného texelu ke zdrojům světla. Čímž se zvýší kvalita a rychlost výpočtu osvětlení.



Obrázek 41: Ukázka vytvořeného interiéru v UE4

5.3 Vnitřní zdroje světla

Další součástí této práce bylo vytvoření scény reprezentující podlaží blokové dozorny virtuální jaderné elektrárny. Oproti předešle popsané scéně zde nejsou žádná okna a tudíž jako zdroje osvětlení sloužily pouze zářivky. Ty byly vytvořeny pomocí statických světel typu **Point Light** v kombinaci s vysoce emisivním materiálem na povrchu zářivek.

Obrázek 42: Referenční fotka BD

Obrázek 43: Finální vizualizace BD

6 Materiály

Jako další ze součástí této práce bylo nastudování a popis vlastností materiálů v Unreal Engine 4. Toto téma je velmi důležité, jelikož celkový vizuální vjem výsledné aplikace je z velké části ovlivněn právě materiály, jež jsou aplikovány na modely ve scéně.

6.1 Základní vlastnosti materiálů

Materiál v UE4 je asset, který může být aplikován na určitý mesh, za účelem kontroly vizuálního vzhledu daného objektu. Technicky řečeno, materiál je použit pro vypočítání výsledné barvy interakce okolních světel s povrchem objektu, na který byl tento materiál aplikován. Tyto výpočty jsou realizovány za pomoci vstupních dat (např. textur), parametrů materiálu a různých matematických operací nad těmito daty. Zjednodušeně řečeno, materiál určuje jaký typ povrchu daný objekt má, jak je reflektivní, jakou má barvu, atd.

Materiály mají mnoho konfigurovatelných vlastností, které určují chování a celkový vzhled materiálu. Zde je seznam základních tří:

6.1.1 Material domain

Tato vlastnost určuje, jak bude daný materiál používán. Je velmi důležitá, jelikož určité typy materiálu (například decals) vyžadují další instrukce navíc pro renderovací engine.

- **Surface** – toto nastavení definuje materiál jako povrch nějakého objektu. (Kovy, plasty, kameny, atd.) – používá se pro drtivou většinu materiálů
- **Deffered decal** – používá se pro tzv. decaly. Decal je v podstatě jednoduchá stěna (plane) promítnuta na určitou část scény (např. zeď) za účelem obohacení scény o další detaily (např. mech na zdi).
- **Light function** – používá se pro speciálně pro světla. Jednoduchým příkladem může býti světlo typu reflektor, které je maskováno určitou texturou
- **Post process** – používá se pro post-process efekty. Neboli efekty, které se aplikují na vyrenderovaný obraz (např. bloom, ssao, vignetting, atd.)

6.1.2 Blend mode

Tato vlastnost určuje, jak bude renderovací engine při renderování kombinovat tento materiál (Source color) s barvou která již existuje ve frame bufferu (Destination color).

- **BLEND_Opaque**
  Final color = Source Color
  To znamená, že se materiál vykreslí přes pozadí. Tento mód je kompatibilní s osvětlením.
- **BLEND_Masked**
  Final Color = Source Color if OpacityMask > OpacityMaskClipValue, else FinalColor = DestinationColor.
  To znamená, že se materiál vykreslí přes pozadí, jen pokud hodnota parametru Opacity mask je vyšší než OpacityMaskClipValue. Tento mód je kompatibilní s osvětlením.
- **BLEND_Transcluent**
  Final Color = Source Color Opacity + Destination Color (1 - Opacity).
  Používá se pro průhledné objekty. Není kompatibilní s dynamickým osvětlením.
- **BLEND_Additive**
  Final color = Source Color + Destination Color.
  Výstup materiálu se přičte k pozadí. Není kompatibilní s dynamickým osvětlením.
- **BLEND_Modulate**
  Final color = Source Color x Destination Color.
  Výstup materiálu se násobí s pozadím. Není kompatibilní s dynamickým osvětlením.

6.1.3 Shading Model

Tato vlastnost určuje, jak budou vstupy materiálu kombinovány.

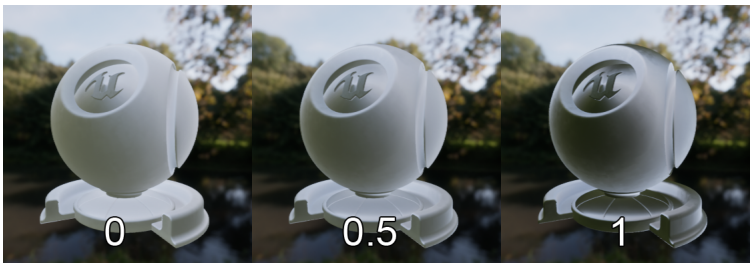
- **Unlit** - Materiál je definován pouze parametry Emmisive a Opacity. Není ovlivněn osvětlením.
- **Default lit** – Výchozí stínovací model. Perfektní pro většinu objektů.
- **Subsurface** – Používán pro materiály využívající subsurface scattering (např. led). Aktivuje parametr Subsurface Color.
- **Preintegrated skin** – Používán pro lidskou kůži. Aktivuje parametr Subsurface Color.
- **Clear coat** – Používán např. pro karosérie aut. Aktivuje parametry Clear Coat a Clear Coat Roughness.
- **Subsurface profile** – Používán pro materiály podobné lidské kůži. Aktivuje parametr Subsurface Color.

6.2 Fyzikální model stínování - základní parametry materiálů

Jako většina moderních enginů,UE4 využívá fyzikálně založené modely a aproximace pro výpočet osvětlení. Což v praxi znamená více fotorealistické výsledky a také odlišný způsob pohledu na strukturu vstupních dat a parametrů pro materiály oproti klasickým metodám. Základem jsou čtyři parametry: base color, metallic, roughness, specular.

6.2.1 Metallic [float]

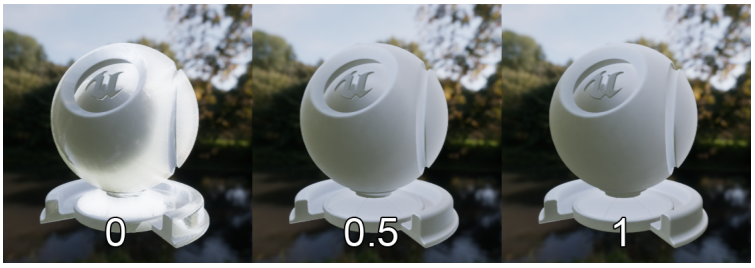
Jedním z hlavních pilířů fyzikálního modelu osvětlení je rozdělení na dvě skupiny: kovy a nekovy. Tyto dvě skupiny se mezi sebou liší způsobem odrazu světelných paprsků. Z fyzikálního hlediska se u nekovů paprsky lomí třemi způsoby: na povrchu tělesa (částečná reflexe okolí), uvnitř tělesa (diffuse scattering) nebo projdou skrz (prosvítání tělesa). Naopak u kovů se téměř všechny paprsky lomí přímo na povrchu tělesa (vzniká zrcadlová reflexe okolí). Tuto vysokou odrazivost kovů způsobuje velké množství volných elektronů na jejich povrchu, které vytvářejí elektromagnetické pole. Aby se určilo, do jaké skupiny daný materiál patří (může být i směsí kovů a nekovů), je nutno nastavit hodnotu parametru „Metallic“ v rozmezí 0 – 1 (1 = 100% kov / 0 = 100% nekov).



Obrázek 44: Parametr Metallic

6.2.2 Roughness [float]

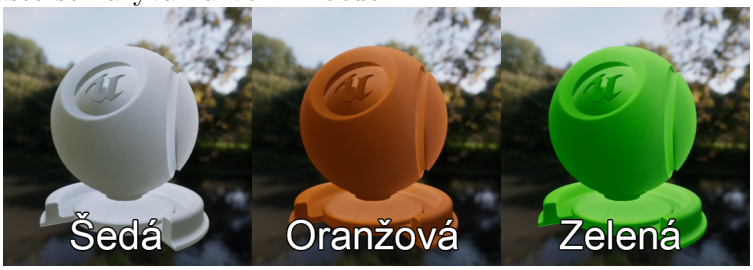
Dalším z důležitých parametrů materiálu je parametr „Roughness“. Je to opět parametr v rozmezí 0 – 1 a slouží k rozlišení mezi hladkými povrchy (Roughness = 0) nebo velmi drsnými povrchy (Roughness = 1). Drsnost povrchu velkou mírou ovlivňuje směr odrazu přicházejících paprsků světla a tudíž celkovou reflexi okolního prostředí.



Obrázek 45: Parametr Roughness

6.2.3 Base color [Vector3]

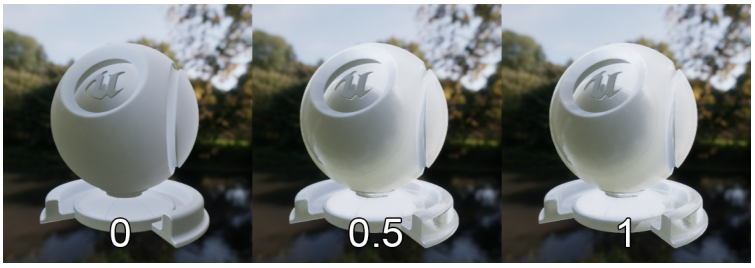
Parametr „Base color“ je jednoduše řečeno základní barva materiálu, která ovlivňuje vzhled celého objektu. Často se nazývá názvem Albedo.



Obrázek 46: Parametr Base Color

6.2.4 Specular [Vector3]

Je to hodnota v rozmezí 0 – 1 a je používána pro určení hodnoty spekularity pro nekovy. Z fyzikálního hlediska nemá žádný vliv na kovy. Pro většinu případů ji není nutno nastavovat (její výchozí hodnota je 0.5). Slouží k zachycení detailnějšího osvětlení spolu s normálovou mapou, tudíž zachytit stíny (okluze) detailů materiálu (praskliny, atd.). Pro zachycení těchto okluzí je automaticky vygenerována cavity mapa, což je v podstatě AO mapa s velmi krátkou délkou paprsků. Tato mapa je dále násobena hodnotou BaseColor což se ve výsledku rovná výstupu Specular. Dalším využitím je například maskování reflexí či nastavení barvy odlesků u kovů (například u zlata).



Obrázek 47: Parametr Specular

6.3 Ostatní parametry materiálu

Krom předešlých čtyř základních (fyzikálně založených) parametrů, Unreal Engine nabízí další potřebné parametry pro vizualizaci složitějších materiálů. Většina z těchto parametrů může být použita pouze s korektním nastavením materiálu (Blend mode, DX11 teselace ON/OFF, atd.).

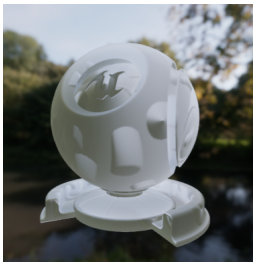
6.3.1 Emmisive [Vector3]

Parametr emmisive simuluje záři materiálu a emitování světla do okolí. Pokud je zapnuto HDR osvětlení, je možno vstupní emmisive barvu násobit skalárem, jež reprezentuje intenzitu záře (ve výsledku bude mít jedna (či více) složek RGB hodnotu vyšší než 1).

Obrázek 48: Parametr Emmisive

6.3.2 Normal [Vector3]

Parametr Normal vyžaduje vstupní texturu „normálovou mapu“. Díky ní mění normálu jednotlivých fragmentů vykreslovaného tělesa a tudíž simuluje různé nerovnosti povrchu, atd.



Obrázek 49: Parametr Normal

6.3.3 Opacity [float]

Parametr v rozmezí 0 – 1 určující průhlednost materiálu. (0 – kompletně průhledné, 1 – kompletně neprůhledné). Použitelné pouze u transkluentních povrchů (Blend mode: transcluent).

Obrázek 50: Parametr Opacity

6.3.4 Refraction [Vector3]

Parametr refraction simuluje index lomu světla (refrakce) skrze transkluentní (průhledné) povrchy jako sklo či voda. Předpokladem funkčnosti je správné nastavení průhlednosti (Opacity).

Obrázek 51: Parametr Refraction

6.3.5 World position offset [Vector3]

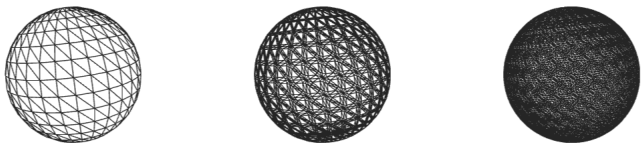
Parametr world position offset simuluje odchylku vrcholů v globálním prostoru zadanou vstupním vektorem. Jinak řečeno: posune vrcholy (vertexy) meshe směrem a vzdáleností, kterou udává vstupní vektor. Samotné posunutí nemusí být konstantní, tudíž je možno vytvořit za pomoci matematických operací (např. sinus, kosinus) různé animace objektů, které jsou proměnné v čase.

Obrázek 52: Parametr World position offset

6.3.6 World displacement, Tesselation multiplier [Vector3, float]

Parametry world displacement a tessalation multiplier jsou spolu úzce spjaty a pro jejich funkčnost je nutno v nastavení materiálu zapnout tesalaci..

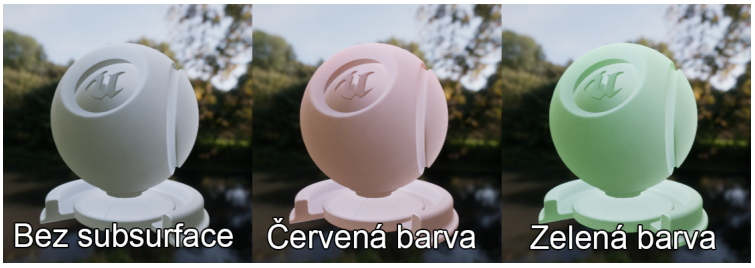
- **World displacement**
  V zásadě funguje obdobným způsobem jako předešle definovaný parametr world position offset (odchýlí vrcholy na základě vstupního vektoru). Ovšem na rozdíl od předchozího parametru, pracuje world displacement pouze s nově teselovanými vrcholy. Při praktickém využití se pro zjištění vstupního (odchylovacího) vektoru používá tzv. height mapa (grayscale textura), která určuje hodnotu odchylky nejbližších (teselovaných) vrcholů v dané části.
- **Tesselation multiplier**
  Určuje množství (kvalitu) teselace na povrchu materiálu. Čím vyšší je hodnota, tím vznikne více nových vrcholů.



Obrázek 53: Tři stupně tesselace

6.3.7 SubSurface Color [Vector3]

Parametr SubSurface color simuluje fyzikální jev, kdy paprsky světla proniknou pod povrch matriálu a roztříští se do všech stran. Díky tomu výsledná barva obsahuje i informaci o barvě pod povrchem. Dobrým příkladem tohoto jevu je lidská kůže, která pod přímým světlem prosvítá červenou barvou (díky krvi pod povrchem). Tudíž v tomto případě by byla hodnota SubSurface Color nastavena na červenou barvu. Pro funkčnost je nutno nastavit v nastavení materiálu atribut „Shading Model“ na „Subsurface".



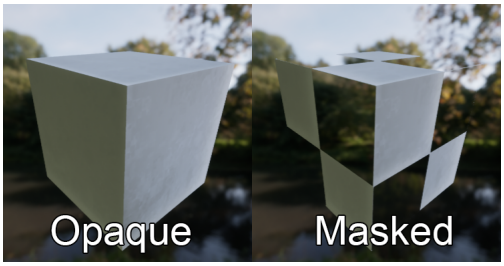
Obrázek 54: Parametr SubSurface Color[2]

6.3.8 Opacity - SubSurface Material [float]

Původně popsaný parametr Opacity se chová odlišným způsobem s materiály, které mají nastaven SubSurface scattering. Na rozdíl od nastavování průhlednosti materiálu, kontroluje hodnotu propustnosti světla roztříštěného pod povrchem tělesa. Na obrázku níže lze vidět, jak nízká hodnota Opacity značí velkou propustnost a naopak vysoká hodnota nízkou propustnost.

6.3.9 Opacity mask [float]

Opacity mask je podobný parametr jako předem definovaný Opacity, avšak s tím rozdílem, že části materiálu jsou buď viditelné (hodnota = 1) nebo neviditelné (hodnota < 1). Díky tomu materiál zachovává vlastnosti osvětlení opaque materiálů (např. přijímá stíny ostatních objektů) a tudíž je vhodným řešením pro komplexní objekty jako zábradlí, drátěné ploty, vegetace atd. Pro správnou funkčnost je nutno nastavit Masked Blend Mode v nastavení materiálu.



Obrázek 55: Standartní opaque (nalevo), cutout se šachovnicovou opacity maskou

6.3.10 Ambient Occlusion [float]

Pomáhá modelům pro výpočet lokálního osvětlení dodat realistický dojem díky tomu, že započítává tlumení světla zastíněním. Ambient occlusion napodobuje skutečné chování světelného záření, zvláště na plochách, které považujeme za matné. Vstupem je zapečená AO mapa (grayscale textura).

6.4 Praktická implementace složitějších materiálů a efektů

Během tvorby této práce bylo vytvořeno mnoho materiálů a efektů, zde je popis těch složitějších.

6.4.1 Vrstvený materiál

Jedním ze složitějších materiálů byl materiál vnitřní stěny chladících věží, jelikož velkou část pokrývají vodní řasy vzhledem k vysoké vlhkosti. Proto byla vytvořena maskovací grayscale textura určující místa na kterých rostou řasy a místa na kterých jsou betonové panely. Pomocí této masky byly dále modifikovány parametry metallic, roughness a specular, což mělo za důsledek výrazněji viditelné a více realistické odlišení mezi těmito dvěma povrchy. A nakonec obdobně jako u materiálu terénu popisovaného v kapitole 4.2.2, bylo vytvořeno několik stupňů opakování textur (tilingu) textury vodních řas pro účel zamezení viditelného opakování.

Obrázek 56: Referenční fotka

Obrázek 57: Finální vizualizace

6.4.2 Refraktivní a reflexní materiál

Během vizualizace podlaží blokové dozorny bylo nutno vytvořit materiál plastového barelu do dávkovače vody, na kterém byla otestována refrakce spolu s reflexí okolního prostředí. Samotná reflexe prostředí byla zachycena do cubemap textury pomocí objektu **Scene Capture Cube** a následně použita (po smíchání s modrou barvou barelu) jako vstup do slotu **Base Color** výstupu materiálu. Následně byla nastavena hodnota parametru **Opacity** na 0.9, což reprezentovalo částečnou průhlednost a hodnota parametru **Refraction** byla vypočítána pomocí interpolace mezi fixní hodnotou a hodnotou získanou za pomoci uzlu **Fresnel**, která zvýšila hodnotu refrakce v záhybech modelu (fragmentech, jejichž normála se více odchylovala od směru ke kameře).

Obrázek 58: Materiál dávkovače vody

6.4.3 Odraz v zrcadle

Během vizualizace podlaží blokové dozorny (konkrétně interiéru výtahu), bylo nutné vytvořit aproximovaný odraz zrcadla. Prvním otestovaným způsobem bylo jednoduché zapečení vnitřní scény výtahu do cubemapy pomocí **Box Reflection Capture** a vytvoření materiálu zrcadla s nastavenými parametry **Metallic** na 1 a parametrem **Roughness** na 0. Tento způsob však produkoval velmi nekorektní výsledky, zejména při pohledu z různých vzdáleností od povrchu zrcadla, jelikož zachycená scéna v cubemapě byla vytvořena jen z jednoho konkrétního místa. Dalším problémem této metody byla nemožnost aktualizace scény v reálném čase, což na jednu stranu bylo z hlediska optimalizace výhodné, avšak na druhou stranu při situaci kdy se scéna dynamicky mění, působí stálý obraz velmi špatně.

Dalším otestovaným a finálně použitým způsobem bylo použití **Scene Capture 2D**, což je v podstatě další kamera, která dokáže v reálném čase zaznamenávat scénu do textury. Pro aproximovanou správnost odrazů bylo nadále nutno vytvořit blueprint jež se staral o změnu pozice a rotace této další kamery, tak aby zrcadlila transformace hlavní kamery.

Obrázek 59: Kabina výtahu se zrcadlem

6.4.4 Vizualizace planet a atmosféry

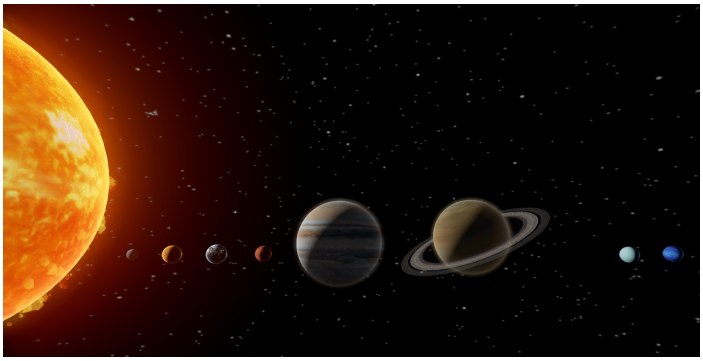
Další ze součástí této práce byla vizualizace sluneční soustavy pro účely edukativní a interaktivní VR aplikace.

Základem bylo vytvoření univerzálního blueprintu planety, který obsahoval dva meshe kulovitého tvaru, kde jeden mesh reprezentoval samotný model planety a druhý její atmosféru. Výjimku tvořil Saturn, jehož součástí byl další model reprezentující prstenec.

Pro každý z těchto modelů byly vytvořeny univerzální materiály. Z těchto univerzálních materiálů byly následně vytvořeny tzv. **Material Instance**, které se využívají pro rychlou tvorbu různých variant materiálů, pomocí změny vstupních parametrů (textur, skalárů či vektorů).

Jak lze vidět z obrázků níže, tak při dostatečném počtu vstupních parametrů lze vytvořit všechny planety sluneční soustavy včetně slunce.

Obrázek 60: Země

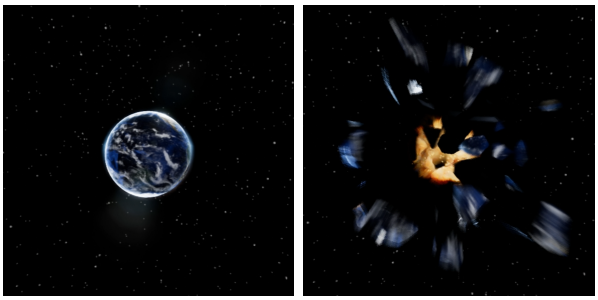


Obrázek 61: Sluneční soustava (nesprávné poměry)

Za účelem interaktivity bylo vytvořeno několik blueprintů. Hlavním blueprintem byl blueprint **Game Manager**, který se staral o veškerou logiku této aplikace. A to přesněji o náhodný výběr určitého pořadí planet, zobrazení tohoto pořadí hráči a následnou detekci zda si hráč toto pořadí správně zapamatoval. Samotné určování zapamatovaných planet bylo realizováno pomocí laserové pistole, kterou hráč ovládal pomocí ručního ovladače. Logika hry byla tedy velmi jednoduchá: Nejprve se zobrazilo pořadí planet, hráč si jej zapamatoval a postupně podle něj planety sestřeloval. Pokud zasáhl planety ve správném pořadí, mohl pokračovat do dalšího kola, které mělo vždy těžší obtížnost (obtížnost se postupně automaticky zvyšovala dvěma parametry: počtu planet v pořadí a času určeného k zobrazení tohoto pořadí planet).

Dále byly vytvořeny dva typy zpětné reakce indikující úspěšnost/neúspěšnost zásahu. Pro indikování správného zásahu byl ke každé planetě přidán další kulovitý model se speciálním materiálem, který byl za účelem postupné animace ovládán z blueprintu. Prvním krokem bylo vytvoření dynamické instance tohoto materiálu za pomocí uzlu **Create Dynamic Material Instance**, což umožňuje dynamickou změnu vstupních parametrů během hry. Dalším krokem bylo vytvoření samotné animace, jež byla vytvořena postupnou inkrementací jednoho skalárního parametru v metodě (eventu) **Tick**, která se volá v konstantních časových intervalech, kde frekvenci těchto volání lze nastavit pomocí parametru **Tick Interval**.

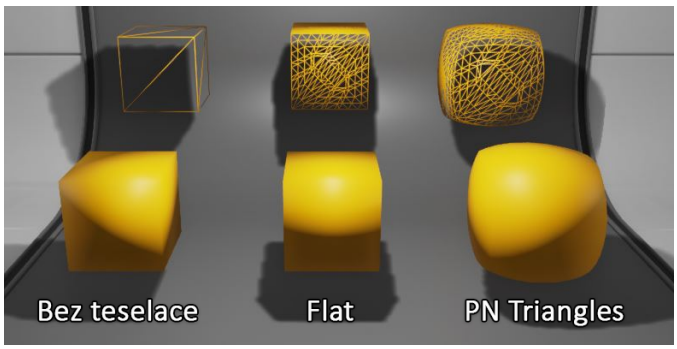
Naopak pro indikaci špatného zásahu byla vytvořena animace výbuchu zasažené planety. Při vytváření tohoto efektu byl nejprve v modelovacím nástroji Blender rozdělen model koule na 50 nepravidelných částí, kde každá z těchto částí byla následně naimportována do projektu jako samostatný mesh. Poté byly všechny tyto části přidány do univerzálního blueprintu planet a při startu hry deaktivovány. Následně v přesný okamžik zásahu byly znovu aktivovány, dále jim byl jednotlivě přiřazen vektor fyzikálního směru a rychlosti (velocity) vypočítaný vždy od středu planety k jednotlivé části. Také jim byla zapnuta fyzikální kolize a simulace (rigidbody). Původně zobrazované modely planety byly rovněž deaktivovány a pro lepší efekt byl přehrán zvuk výbuchu s částicovými efekty. Což mělo za následek rozmetání planety do všech stran.



Obrázek 62: Animace exploze

6.4.5 Teselované materiály

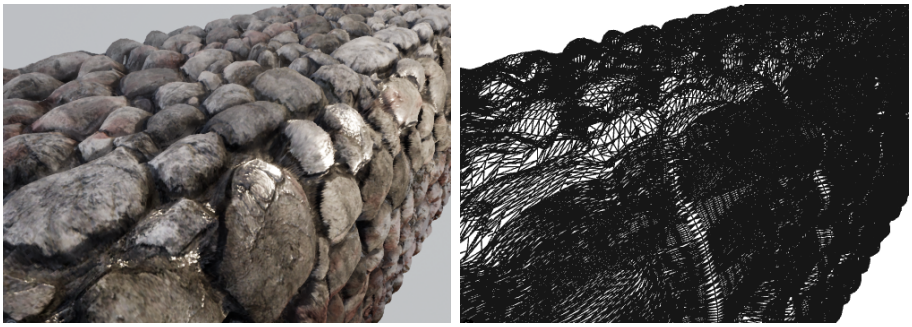
Teselace je technika modifikace topologie objektů pomocí vytváření dalších bodů v topologii 3D modelu. Unreal Engine podporuje dva typy teselace **Flat tesselation** a **PN Triangles**. Kde **Flat** teselace funguje na jednoduchém principu půlení trojúhelníků a **PN Triangles** na principu zakřivených Beziérových trojúhelníků, což v praxi dokáže více zaoblit objekt. [16] Následně se využívá speciální černobílá textura tzv. výšková mapa, která určuje, jak moc se mají vertexy v určitém místě vychýlit od jejich původní pozice, čímž vznikne detailní tvarování nerovností.



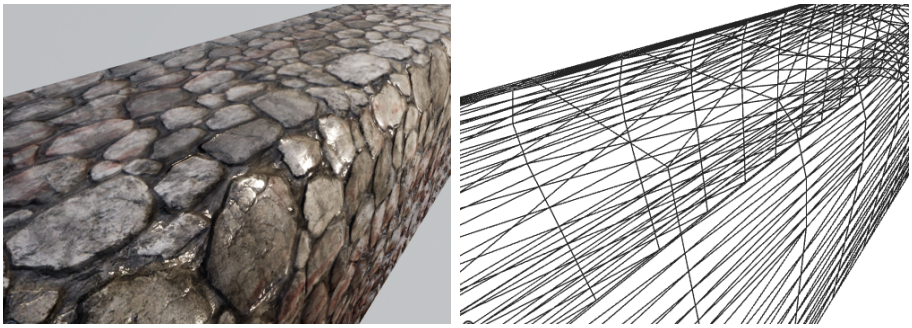
Obrázek 63: Flat a PN Triangles teselace [10]

Pro otestování teselace byl nejprve vytvořen jednoduchý model zdi. Následně bylo za potřebí vytvořit materiál s nastavenou teselací na **Flat teselation** a vytvořeným výpočtem interpolace mezi maximálními a minimálními hodnotami parametrů **Tesselation multiplier** a **World Displacement** na základě vzdálenosti vykreslované části zdi od kamery (vzdálenost se počítala pomocí uzlů **Absolute World Position** a **Camera Position WS**). Takže pokud se kamera přibližovala ke zdi, hodnoty parametrů se přibližovali k nastavenému maximu a naopak pokud se oddalovala, tak se hodnoty přibližovali k minimu. Důvod, proč bylo potřeba měnit tyto parametry na základě vzdálenosti od kamery, byl jednoduchý. Jelikož v situaci kdy je vykreslovaný objekt v dálce, není nutné jej teselovat a díky tomu se ušetří výkon GPU.

Na závěr je ještě nutno podotknout, že aby bylo docíleno požadované změny tvaru zdi, bylo nutno parametr **World Displacement** násobit hodnotami získanými z černobíle výškové mapy, která určovala právě tvar výsledného teselovaného modelu.



Obrázek 64: Zeď s teselací



Obrázek 65: Zeď bez teselace

7 Závěr

Během realizace této práce bylo otestováno mnoho funkčních prvků Unreal Engine a zároveň bylo vytvořeno 7 aplikací, včetně 2 hratelných VR her. Každá z těchto aplikací demonstrovala jednotlivé možnosti Unreal Engine jako: vizualizace interiéru, vizualizace terénu, simulace látky, simulace vlasů, interaktivní vizualizace sluneční soustavy, animování 3D modelů postav, vytvoření střelecké VR hry se 4 druhy zbraní, atd.

Také bylo z části přispěno do projektu vizualizace virtuální jaderné elektrárny (scéna podlaží blokové dozorny, chladící věže, terén a prostředí venkovní mapy, interaktivní tlačítka pro VR ovladače, materiály budov, atd.). Přičemž byla vyzkoušena práce v týmu, kde každý ze studentů implementoval specifické scény a funkce. Což přineslo i pohled na odlišné oblasti tohoto enginu, jež zpracovávali jiní kolegové.

Další zajímavou zkušeností byla prezentace VR aplikací během dne otevřených dveří na VŠB-FEI, což poskytlo zpětnou odezvu od testujících návštěvníků, která byla následně vzata v potaz při vylepšování/opravy částí této aplikace.

Literatura

[1] SKALSKÝ, Michal. *Eclipse engine*. Plzeň, 2010. Diplomová práce. Západočeská univerzita v Plzni.

[2] *Unreal Engine*. [online]. [cit. 2017-03-14]. Dostupné z: `https://www.unrealengine.com/unreal-engine-4`

[3] *Unity*. [online]. [cit. 2017-03-14]. Dostupné z: `https://unity3d.com/unity/engine-features`

[4] *CryEngine*. [online]. [cit. 2017-03-14]. Dostupné z: `https://www.cryengine.com/features`

[5] What game engines do you currently use? In: *The Next Web* [online]. 2016 [cit. 2017-03-09]. Dostupné z: `https://thenextweb.com/gaming/2016/03/24/engine-dominating-gaming-industry-right-now/`

[6] C. ÖZER, M. Skinning. In: *Skinning | M. Cihan ÖZER* [online]. 2016 [cit. 2017-03-11]. Dostupné z: `http://www.mcihanozer.com/tips/computer-graphics/skinning/`

[7] Inverse Kinematics. , funk.*Kinematics & Dynamics* PRINCETON UNIVERSITY, Department of COMPUTER SCIENCE [cit. 2017-04-17]. Dostupné z: `http://www.cs.princeton.edu/courses/archive/fall00/cs426/lectures/kinematics/img008.gif`

[8] *Landscape Technical Guide*. [online]. Epic Games [cit. 2017-03-14]. Dostupné z: `https://docs.unrealengine.com/latest/INT/Engine/Landscape/TechnicalGuide/`

[9] *World Machine* [online]. [cit. 2017-03-27]. Dostupné z: `http://www.world-machine.com/`

[10] *Unreal Engine 4 Documentation* [online]. Epic Games [cit. 2017-04-02]. Dostupné z: `https://docs.unrealengine.com/latest/INT/`

[11] Paragon Feature Examples: Foliage & Parallax Occlusion Mapping | Feature Highlight | Unreal Engine. In: *YouTube* [online]. Unreal Engine [cit. 2017-04-11]. Dostupné z: https://www.youtube.com/watch?v=4gBAOB7b5Mg

[12] *NVIDIA HairWorks* [online]. NVIDIA [cit. 2017-04-02]. Dostupné z: `https://developer.nvidia.com/hairworks`

[13] *MakeHuman* [online]. The MakeHuman team. [cit. 2017-04-02]. Dostupné z: `http://www.makehuman.org/`

[14] *NVIDIA GameWorks and UE4* [online]. NVIDIA [cit. 2017-04-02]. Dostupné z: `https://developer.nvidia.com/nvidia-gameworks-and-ue4`

[15] NVIDIA APEX. *NVIDIA* [online]. [cit. 2017-04-18]. Dostupné z: http://www.nvidia.com/object/apex.html

[16] VLACHOS, Alex, Jorg PETERS, Chas BOYD a Jason L. MITCHELL. *Curved PN Triangles* [online]. [cit. 2017-04-21]. Dostupné z: http://alex.vlachos.com/graphics/CurvedPNTriangles.pdf

[17] Shading Language used in Unity. *Unity | DOCUMENTATION* [online]. [cit. 2017-04-24]. Dostupné z: https://docs.unity3d.com/Manual/SL-ShadingLanguage.html

[18] *Custom Expressions*. [online]. Epic Games [cit. 2017-03-14]. Dostupné z: https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/ExpressionReference/Custom/

A Příloha - CD s ukázkovými projekty

V přiloženém CD se nachází první část ukázkových aplikací, včetně textového souboru s odkazem na stažení druhé části. V této první části jsou tedy tyto spustitelné aplikace:

- **Cloth** - ukázka fyziky látky
- **Terén** - ukázka složitého materiálu terénu (se stupni opakování textur)
- **UELightmap** - názorná ukázka nejčastějších chyb nastávajících při zapékání lightmap
- **VR Planety** - jednoduchá hratelná VR hra ve které je vytvořená sluneční soustava

Ve druhé části, jež byla nahrána na externí zdroj na internetu z důvodu nedostatečně velké přidělené kapacity, lze nalézt tyto aplikace

- **HairWorks** - ukázka fyziky vlasů a jednoduchých animací emocí lidského obličeje
- **VR Interier** - vizualizace interiéru, ve které lze ozkoušet i jednoduchou minihru zapojování logických obvodů
- **Zombie VR hra + testování inverzní kinematiky** - hratelný VR projekt, ve které je nutné se bránít hordám zombie za pomocí 4 druhů zbraní. Je vhodné jej spouštět ze zdrojového projektu přímo v Unreal Engine, jelikož je jeho součástí i level obsahující aproximovanou animaci VR hráče za pomoci ovladačů virtuálního headsetu. Více informací je v přibaleném textovém souboru readme.
- **Zdrojové projekty všech aplikací**